KATHOLIEKE
UNIVERSITEIT
LEUVEN

DEPARTEMENT TOEGEPASTE ECONOMISCHE WETENSCHAPPEN

RESEARCH REPORT 9937

DISTANCE: A FRAMEWORK FOR SOFTWARE MEASURE CONSTRUCTION

by

G. POELS

G. DEDENE

D/1999/2376/37

DISTANCE: A framework for software measure construction

Geert Poels and Guido Dedene

Management Information Systems Group
Dept. Applied Economic Sciences
Katholieke Universiteit Leuven
Naamsestraat 69, B-3000 Leuven, Belgium
{geert.poels, guido.dedene}@econ.kuleuven.ac.be

Abstract

In this paper we present a framework for software measurement that is specifically suited to satisfy the measurement needs of empirical software engineering research. The framework offers an approach to measurement that builds upon the easily imagined, detected and visualised concepts of similarity and dissimilarity between software entities. These concepts are used both to model the software attributes of interest and to define the corresponding software measures. Central to the framework is a process model that embeds constructive procedures for attribute modelling and measure construction into a goal-oriented approach to empirical software engineering studies. The underlying measurement theoretic principles of our approach ensure the construct validity of the resulting measures.
The approach was tested on a popular suite of object-oriented design measures. We further show that our measure construction method compares favourably to related work.

Index Terms

Empirical software engineering, software measurement, measure validation, measurement process models, software attribute modelling

1 INTRODUCTION

Software engineering is the study of tools, methods and processes for the efficient and effective development, implementation and maintenance of software. Software engineering researchers have come to realise that a substantial body of empirical evidence is needed to support the claims of novel technologies. The last couple of years saw a tremendous increase in empirical validation studies, particularly with respect to object-oriented technologies [6]. However, there is a significant need for more empirical research in software engineering [16]. Moreover, a number of meta-analyses and state-of-the-art reviews have identified serious problems with the current state of practice in empirical software engineering [10], [31], [36], [56].

Most of the critiques concern the internal validity of empirical studies. An internally valid study is one that has been carried out correctly, such that we can have confidence in the study results. Threats to the internal validity are mostly caused by flaws in the experimental design (e.g., little control of confounding effects), the use of inappropriate analysis techniques (e.g., a deviation from the assumed underlying data distribution), or the misinterpretation of results (e.g., statistical insignificance of results).

Another area of concern is the external validity of empirical software engineering studies. The study results need to be confirmed by internal and external replication experiments in order to be confident in their generalisation. Currently, the replication of experiments is an exception

rather than a rule in the software engineering field. A notable exception is the experiment of Daly *et al.* [24], investigating the effect of inheritance depth on the maintainability of object-oriented software, that was replicated by Cartwright *et al.* [17] and Harrison *et al.* [33].

There is at least one other aspect of validity that is addressed by the critiques on contemporary empirical software engineering studies, i.e., the construct validity of the measures for the dependent and independent variables. A measure exhibits construct validity when it measures what it is purported to measure. Clearly, construct validity is a prerequisite for both the internal and external validity of a study. Problems with this type of validity result in other variables being measured than the ones originally intended. As a consequence, conclusions may be drawn from an analysis of measurement values that do not relate to the original hypothesis being tested.

There seems to be consensus among software measurement scientists that construct validity must be evaluated with respect to measurement theory [10], [29], [59]. Although the role of measurement theory in the construction of measurement instruments for software engineering research has been acknowledged since at least ten years (see e.g., [1], [60]), there are still many open problems regarding the application of the theory. Many of the proposals for a measurement theoretic approach to software measurement focus on measure properties (e.g., scale type) and pay insufficient attention to constructive aspects, i.e., modelling and measurement procedures. Moreover, few of the proposals qualify as systematic approaches as they offer little practical guidance regarding their application, for instance by means of a process model. Finally, software measurement scientists have barely scratched the surface of measurement theory. The measurement structures that have actually been used for software measurement are largely confined to ordinal representations [27], belief function structures [58], and extensive measurement [59]. However, as shown in [55], there is a wealth of other measurement structures that need to be investigated yet, in particular with respect to some types of software, like object-oriented software, where the 'classic' measurement structures fail or lead to sub-optimal results.

In this paper we present a framework for software measure construction that addresses the issues raised above. The framework provides constructive procedures to model the software attributes of interest and define the corresponding measures. The different procedure steps are inserted into a process model for software measurement that (i) details for each task the required inputs, underlying assumptions and expected results, (ii) prescribes the order of execution, providing for iterative feedback cycles, and (iii) embeds the measurement procedures into a typical goal-oriented measurement approach such as, for instance, GQM [4] or GQM/AF [43].

The framework is called DISTANCE as it builds upon the concepts of distance and dissimilarity, (i.e., conceptual distance). Software attributes are modelled as (conceptual) distances between the software entities they characterise and other software entities that serve as reference points or norms for measurement. These distances are then measured by mathematical functions that satisfy the axiom set of the metric space. Although Kitchenham *et al.* [37] contend that the use of axioms (in a mathematical sense) for software measurement is premature, they state that "a study of this axiom set might be valuable to software metrics researchers" (p. 281). The approach we present in this paper is a result of studying the metric axioms.

Working with (conceptual) distances offers many advantages as the concepts of similarity and dissimilarity are well-understood, easily imagined and used in every day life. They are also

flexible concepts as they require an explicit context of evaluation, i.e., entities are similar or dissimilar with respect to specific characteristic(s). Last but not least, dissimilarity is a concept that has been formally defined in measurement theory by means of proximity structures [52]. Measuring distance or dissimilarity thus amounts to the construction of a proximity representation, for which definitions and guidelines can be found in textbooks on measurement theory.

On the one hand, care has been taken to build the measurement theoretic interpretation of dissimilarity into our framework. This ensures that that the construct validity of the measures obtained with DISTANCE is guaranteed, i.e., it can be formally proven that a proximity representation is constructed. On the other hand, the constructive attribute modelling and measure definition procedures as presented in the process model hide the complexity of the underlying measurement theoretic constructs from the user. We take full advantage of the intuitiveness and flexibility of the distance concept to arrive at a measure construction framework that is transparent with respect to measurement theory and that is generic, i.e., not limited to the measurement of a specific software attribute.

The arguments made in this paper are not meant to claim that measure validity is more important than the other aspects to consider with regard to empirical software engineering studies (e.g., experimental design, statistical power). Neither is it our intent to discourage researchers from performing empirical research altogether. Nevertheless, we hope that the DISTANCE framework provides part of an answer to the complex problems faced by contemporary empirical software engineering research.

The organisation of this paper is the following. In section 2 we review related work in the field of formal software measurement. In section 3 the DISTANCE framework is presented. First, an overview of DISTANCE is provided by means of a process model for measure construction. Next, the individual tasks in the attribute modelling and measure definition procedures are discussed in more detail. Section 4 evaluates distance-based measurement from a measurement theoretical perspective. We map the concepts of our approach into measurement theoretical concepts and show how the construct validity of the measures can be proven. Distance-based measurement must also be evaluated from a practical perspective. Section 5 contains a case study relating to object-oriented software measurement as this is a field where several problems with construct validity and the application of measurement theory to software measurement have been noted [35], [58]. In the case study the well-known MOOSE measures (Metrics for Object Oriented Software Engineering) of Chidamber *et al.* [19], [20] are redefined using DISTANCE. The result of this exercise is a suite of object-oriented design measures with proven construct validity. Finally, conclusions are presented in section 6.

2 RELATED WORK

We first present some terminology and definitions from measurement theory. Next, related work is discussed.

2.1 Measurement Theory

According to measurement theory, a measure is a homomorphism from an empirical relational system into a numerical relational system. A relational system is an ordered $(1 + p + q)$-tuple $\boldsymbol{A} = (A, R_1, R_2, ..., R_p, o_1, o_2, ..., o_q)$, where A is a set, $R_1, R_2, ..., R_p$ are relations on A and $o_1, o_2, ..., o_q$ are binary operations on A. If $\boldsymbol{A}$ is an empirical relational system, then A is a set of

entities that can be observed in reality. The relations R_1, R_2, ..., R_p order or classify these entities with respect to some attribute. Assume that $\boldsymbol{B} = (B, S_1, S_2, \ldots, S_p, \oplus_1, \oplus_2, \ldots, \oplus_q)$ is a numerical relational system of the same type as $\boldsymbol{A}$, i.e., B is a set of numbers, to each m-ary relation R_i on A corresponds a m-ary relation S_i on B, and to each binary operation o_j on A corresponds a binary operation $\oplus_j$ on B. A function μ: A $\rightarrow$ B is a measure if it is a homomorphism from $\boldsymbol{A}$ into $\boldsymbol{B}$, i.e., $\forall$ $a_1, a_2, \ldots, a_m \in$ A:

$$R_i(a_1, a_2, \ldots, a_m) \Leftrightarrow S_i(\mu(a_1), \mu(a_2), \ldots, \mu(a_m)), \quad i = 1, 2, \ldots, p$$

and $\forall$ a, b $\in$ A:

$$\mu(a\ o_j\ b) = \mu(a) \oplus_j \mu(b), j = 1, 2, \ldots, q$$

If μ is a homomorphism from $\boldsymbol{A}$ into $\boldsymbol{B}$, then the triple $(\boldsymbol{A}, \boldsymbol{B}, \mu)$ is a scale.

A representation theorem states necessary and sufficient conditions for the existence of a homomorphism from a given empirical relational system into some numerical relational system, mostly referring to the set of real numbers. These conditions depend on the level of empirical understanding of the attribute, i.e., the sophistication of the empirical relational system.

In the simplest case the attribute of interest allows to distinguish the observed entities, but not to order them. This means that empirically, the set of observed entities is partitioned into a number of equivalence classes. For instance, cars can be classified based on their colour. However, there is no implicit ordering in the set of colours. The only conclusion that can be drawn based on colour is that two cars have the same colour or have a different colour. The relation 'has the same colour as' is an equivalence relation. It is reflexive, symmetric and transitive. The assignment of colour labels to cars is not measurement in the strict sense, since a mapping is made into a relational system that is not numerical. However, the colour label (e.g., green, red, yellow, ...) assigned to cars is the result of a homomorphism from ({cars}, 'has the same colour as') into ({colour labels}, =). For colour: {cars} $\rightarrow$ {colour labels} it holds that $\forall$ A, B in {cars}:

$$\text{A has the same colour as B} \Leftrightarrow \text{colour(A)} = \text{colour(B)}$$

A higher level of sophistication is observed when the attribute orders the set of entities. In this case the empirical relational system contains at least one ordering relation on the set of entities. For instance, in the empirical relational system (A, R), where A is a set of people and R is the relation 'is not taller than', R orders the persons in A according to their height. The representation theorem of ordinal measurement states that whenever R is binary (i.e., R $\subseteq$ A $\times$ A), then there is a function μ: A $\rightarrow \Re$ that satisfies $\forall$ a, b $\in$ A:

$$(a,b) \in R \Leftrightarrow \mu(a) \leq \mu(b)$$

if and only if R is a weak order (i.e., transitive and strongly complete).

The triple ((A, R), ($\Re$, $\leq$), μ) is an ordinal scale. Some books on measurement theory (see e.g., [38]) provide guidelines for the construction of a function satisfying this representation condition.

A level of sophistication that is sufficiently high for most practical uses of measurement is associated with the representation theorem of extensive measurement. This representation

theorem requires that the empirical relational system has the form (A, R, o), where A is a set of entities, $R \subseteq A \times A$ is an ordering relation and o is a binary operation on A. There is a function $\mu: A \rightarrow \Re$ such that $\forall$ a, b $\in$ A:

$$(a,b) \in R \Leftrightarrow \mu(a) \leq \mu(b)$$

and

$$\mu(a \circ b) = \mu(a) + \mu(b)$$

if and only if (A, R, o) is an extensive structure.

The empirical relational system (A, R, o) is an extensive structure if the following axioms are satisfied:

1) (A, R) is a weak order,

2) $\forall$ a, b, c $\in$ A: $a \circ (b \circ c) \approx (a \circ b) \circ c$, *(weak associativity)*

3) $\forall$ a, b, c $\in$ A: $(a,b) \in R \Leftrightarrow ((a \circ c), (b \circ c)) \in R \Leftrightarrow ((c \circ a), (c \circ b)) \in R$, *(monotonicity)*

4) $\forall$ a, b, c, d $\in$ A: if $(a,b) \in R_S$ then for any c, d there exists a natural number n, such that $((n.a \circ c), (n.b \circ d)) \in R$, *(Archimedian axiom)*

where

$a \approx b \Leftrightarrow (a,b) \in R$ and $(b,a) \in R$

$(a,b) \in R_S \Leftrightarrow (a,b) \in R$ and $(b,a) \notin R$

$n.a = a \circ a \circ a \circ \ldots \circ a$ (n times a)

2.2 Measurement Theoretic Approaches to Software Measurement

The GRUBSTAKE approach of Melton *et al.* [42] and the Model-Order-Mapping (MOM) approach of Gustafson *et al.* [32] contain a constructive procedure to define a partial order on a set of software abstractions. This order refers to empirical statements regarding the attribute of interest that express a certain level of consensus or common understanding. An example of such a statement might be '*adding an edge to a flow-graph cannot make the flow-graph less complex*'. The set of software abstractions together with the empirical order that was constructed builds the empirical relational system that models the attribute of interest (e.g., control-flow complexity). GRUBSTAKE and MOM do not provide a procedure to define a measure for the attribute of interest. However, they require that such a measure must preserve the empirical order on the set of software abstractions, i.e., if (A, R) is the empirical relational system and (B, S) is the numerical relational system then it is required that a measure $\mu: A \rightarrow B$ satisfies $\forall$ a, b $\in$ A: $(a,b) \in R \Rightarrow (\mu(a),\mu(b)) \in S$.

To be a measure in the sense of measurement theory (i.e., a homomorphism from (A, R) into (B, S)), μ should also satisfy $\forall$ a, b $\in$ A: $(\mu(a),\mu(b)) \in S \Rightarrow (a,b) \in R$. Fenton [28] notes that this requirement is problematic for GRUBSTAKE when the function μ is real-valued. The empirical order R (e.g., *is less complex than*) is not strongly complete, and thus the requirements for a homomorphism into a numerical relational system $(\Re, <)$ are not met. So, if the function $\mu: A \rightarrow \Re$ is used as a measure in the sense of measurement theory then an ordering might be forced upon software abstractions that are incomparable with respect to the attribute of interest.

In [27] Fenton considers using weak ordering relations (instead of partial orders) to model software attributes. If the empirical relational system consists of a set of empirical objects and a weak ordering relation on this set, then the ordinal representation theorem of measurement theory applies. Procedures to construct the weak empirical order and the measure are not discussed in [27].

Zuse *et al.* [60] present a formal approach to validate software measures. They developed a procedure to describe the weak empirical order that is assumed by a proposed measure. Their formal approach further checks whether the proposed measure is additive relative to concatenation operations on the empirical objects. Such an additive measure assumes the software attribute to be described by an extensive structure. The validation of a measure consists of checking whether the empirical relational structure assumed by the proposed measure corresponds to the empirical relational structure that describes a person's understanding of the attribute of interest [59].

In [58] Zuse showed that many measures proposed for object-oriented software do not assume an extensive structure. Many measures are not additive relative to intuitive concatenation operations for object-oriented software such as class unification. It was however demonstrated that these object-oriented software measures satisfy the axioms of the modified function of belief.[1] Such measures assume the attribute to be modelled by a modified relation of belief.[2]

Whitmire proposes in [55] measures for a range of attributes of object-oriented designs. For each attribute a suitable model (i.e., empirical relational structure) is chosen and the appropriate representation theorem is applied. For some attributes alternative models are presented. The approach of Whitmire is more explorative than constructive. On the one hand this offers the advantage that the model can be chosen that matches most closely a person's understanding of the software attribute. On the other hand we believe that a more systematic approach is called for.

2.3 Property-Based Approaches to Software Measurement

A number of approaches have proposed sets of desirable properties for software measures [13], [25], [39], [44], [53], [54]. Existing or newly proposed measures are verified with respect to these properties. Measures that do not satisfy one or more of the properties are rejected.

Briand *et al.* [13] have clarified the relationship between property-based approaches and measurement theory. They argue that desirable measure properties must be seen as properties that characterise the numerical relational system. Instead of explicitly defining the empirical relational system, the property-based approaches define the properties of the numerical relational system that are preserved from corresponding properties of the empirical relational system. However, these empirical properties do not necessarily correspond to the (necessary and sufficient) empirical conditions required by the representation theorems of measurement theory. For instance, in [48] it was shown that the empirical properties that a measure must preserve (as desired by a property-based approach) only describe a partial empirical order. As a

[1] The modified function of belief is derived from Dempster and Shafer's theory of belief functions. A belief is a concept that generalises the concept of probability. A function of belief describes the belief in a hypothesis and the belief in a combination of hypotheses [59].

[2] A modified relation of belief is a binary relation on the set of empirical objects satisfying the axioms of the weak order and a number of additional axioms (i.e., dominance, partial monotonicity, weak positivity) that describe the effect of concatenation on the empirical weak order [59].

consequence, the sets of desirable measure properties proposed by the property-based approaches are generally not sufficient to prove the construct validity of measures.

In [14] Briand *et al.* make a distinction between generic properties and context-dependent properties. The former correspond to the desirable measure properties as proposed in, for instance, [13] and [44]. On their own they are not sufficient to validate a measure. The latter type of properties is used to define a total order (e.g., a weak order) on the set of empirical objects. A context-dependent property is similar to the representation condition of ordinal measurement and thus guarantees the existence of a homomorphism. The generic properties may further increase the sophistication of the empirical relational system, for instance, by requiring a measure to be additive relative to a concatenation operation.

2.4 Evaluation of Related Work

Table 1 evaluates the measurement theoretic and property-based approaches with respect to a number of criteria. These correspond to desirable features of a software measurement approach, mainly from the perspective of empirical software engineering research (cf. introduction). We briefly discuss these criteria below.

Criterion 1: *Is the approach sufficient to prove the construct validity of a measure?*

This is of course the main evaluation criterion given our focus on construct validity. We require that a measurement approach is consistent with measurement theory, as this is the reference framework for measure validity.

As shown by Fenton [28], the GRUBSTAKE and MOM approaches do not satisfy Criterion 1. The same problem has been noted for property-based approaches [48]. The other approaches in Table 1 all allow for some kind of formal check or proof of a measure's construct validity.

Criterion 2: *What type of scale is supported by the approach?*

The scale type refers to the uniqueness of a scale ($\boldsymbol{A}$, $\boldsymbol{B}$, μ), i.e., how narrow or broad is the class of scale transformations that result in other homomorphic mappings from $\boldsymbol{A}$ into $\boldsymbol{B}$? Some well-known scale types, along with the class of admissible transformations of scale they describe, ranging from broad to narrow (i.e., from low to high uniqueness), are:

- Ordinal — The class of monotone increasing transformations
- Interval — The class of positive linear transformations
- Ratio — The class of similarity transformations
- Absolute — The identity transformation

Scale types are useful to discriminate meaningful and meaningless statements involving measurements. A statement involving measurements is meaningful if its truth is invariant to the admissible transformations of scale. It has been shown that many of the descriptive statistics (e.g., arithmetic mean, variance) used by parametric statistical techniques are not meaningful for scale types lower than interval, or even ratio [27]. As a consequence, it has been advised to use only non-parametric statistics (e.g., Spearman's rank correlation) when analysing ordinal measurement data. However, Briand *et al.* [11] have shown that many parametric statistical techniques are quite robust for deviations of the interval or ratio scale type. They argue that these techniques, more powerful than non-parametric techniques, can be used for ordinal data. It must be acknowledged that Briand *et al.* base their arguments on experiments conducted in the social sciences. The effect of using parametric statistical techniques for the analysis of

ordinal software measurements has not been examined so far. Therefore, we claim that it is better to know the scale type of a proposed scale and that we should strive for scale types that reflect our empirical understanding of a software attribute.

The concept of scale type is closely related to the sophistication of an empirical relational system. Many representation theorems in measurement theory are accompanied by a uniqueness theorem stating the scale type of the representation. For those approaches in Table 1 that are consistent with measurement theory we can therefore examine the scale types that are supported. Note that the belief function approach in [58] results in scales characterised by an unnamed scale type, higher than ordinal but lower than ratio.

We agree with Zuse [59] that the ordinal level is not sufficient for software measurement in general and with Morasca *et al.* [45] that often (but not necessarily always) the empirical understanding of a software attribute goes further than a mere ordinal classification. Therefore, the approach of Fenton as described in [27] is not sufficient from the perspective of empirical software engineering research.

Criterion 3: *Does the approach propose a constructive procedure to formalise the empirical understanding of the attribute of interest by means of an empirical relational system?*

The definition of a measure (in the sense of measurement theory) requires that first the attribute of interest is described by means of an empirical relational system. We require that a measurement approach provides a constructive procedure to define such empirical relational systems.

As can be observed in Table 1, the evaluation of Criterion 3 results in a rough distinction between approaches that aim at the construction of new software measures and those that aim at the validation of existing measures.[3] In particular, the approach of Zuse [58], [60] and the property-based approaches examine the construct validity of existing software measures. Being consistent with measurement theory, Zuse's approach provides in a procedure to describe the empirical relational system assumed by a measure. However, it does not result in such a description prior to the existence of a software measure. Whereas hundreds of software measures have been presented in the literature [59], it cannot be ascertained yet that a suitable candidate measure can be found for all measurement needs. From the perspective of empirical software engineering research, a measurement approach must allow for the definition of new measures whenever such a need is felt.

The GQM/MEDEA approach of Briand *et al.* [14] is a property-based approach that is not restricted to the validation of existing measures. The context-dependent properties can be used to derive empirical relational systems, which may be refined using generic properties (cf. supra).

Criterion 4: *Does the approach propose a constructive procedure to define a homomorphic mapping (i.e., a measure) into a numerical relational system?*

Measurement theory textbooks (see e.g., [38], [49]) provide constructive procedures to construct homomorphic mappings once an empirical relational system has been defined. From the approaches in Table 1, only Whitmire [55] deals with the topic of measure construction. It

[3] With the exception of [27] which does not particularly aim at validating existing software measures (cf. supra).

seems that this final step is generally missing in the related work that was reviewed.[4] We require it explicitly from a formal measurement approach.

Criterion 5: *Is it a systematic approach that offers practical guidance by means of a process model for measure construction?*

The constructive procedures for the formalisation of the attribute of interest and the construction of a measure function must be inserted into a process model for measure construction (cf. introduction). A basic process model has been used to describe the MOM approach [32]. Hitz *et al.* [35] have presented another process model for measure construction. In neither case is the measure construction process embedded in the global framework of empirical software engineering research. The explorative character of Whitmire's approach [55] has been noted before (cf. supra).

The only approach that offers a detailed and fully integrated process model for measure construction is GQM/MEDEA [14]. However, GQM/MEDEA does not state how to define a homomorphic mapping once a software attribute has been formalised by generic and context-dependent properties (Criteria 4). As a consequence, it is not exactly clear how the approach relates to the representation theorems found in measurement theory, other than that of ordinal measurement. Nevertheless, of all reviewed approaches, GQM/MEDEA is most closely related to the DISTANCE framework presented in the next section when evaluated from the perspective of empirical software engineering research.

3 THE DISTANCE FRAMEWORK FOR SOFTWARE MEASURE CONSTRUCTION

Central to the framework is the concept of distance as defined in mathematics. Actually, no definition for distance, as a concept, is found in mathematics. However, the concept of a distance function on a set has been given an axiomatic definition. Such a definition consists of a set of axioms that are both necessary and sufficient. The axioms that define a distance function are called the metric axioms. A distance function is also called a metric. A set and a metric defined on that set are collectively referred to as a metric space. It must be noted that in the software measurement literature the terms 'metric' and 'software metric' are used in a more general sense, i.e., to denote "any number extracted from a software entity" ([30], p. 324). In the DISTANCE framework, the term 'metric' is properly used.

DEFINITION 1 (Metric and Metric Space)

(1) Let A be a set. The function $\delta: A \times A \rightarrow \Re$ is a metric if and only if:

A_1. $\forall a, b \in A: \delta(a, b) \geq 0$	*(non-negativity)*
A_2. $\forall a, b \in A: \delta(a, b) = 0 \Leftrightarrow a = b$	*(identity)*
A_3. $\forall a, b \in A: \delta(a, b) = \delta(b, a)$	*(symmetry)*
A_4. $\forall a, b, c \in A: \delta(a, b) \leq \delta(a, c) + \delta(c, b)$	*(the triangle inequality)*

(2) If $\delta: A \times A \rightarrow \Re$ is a metric, then (A, δ) is a metric space.

[4] Another constructive procedure, based on an assignment of prime numbers, has been presented in [28], but as acknowledged by the author himself, "is of little practical value" ([28], p. 362) and "is of purely theoretical interest" ([29], p. 202).

	SUFFICIENT TO PROVE CONSTRUCT VALIDITY?	WHAT SCALE TYPE IS SUPPORTED ?	PROPOSES A CONSTRUC-TIVE PROCEDURE TO DEFINE EMPIRICAL RELATIONAL SYSTEM?	PROPOSES A CONSTRUC-TIVE PROCEDURE TO DEFINE MEASURES?	PROPOSES A PROCESS MODEL FOR MEASURE CONSTRUC-TION?
Property-based approaches	N	\	N	N	N
GRUBSTAKE [42]	N	\	Y	N	N
MOM [32]	N	\	Y	N	Y
Fenton [27]	Y	O	N	N	N
Zuse and Bollmann [60]	Y	O,I,R	N	N	N
Zuse [58]	Y	>O, <R	N	N	N
Whitmire [55]	Y	All	Y	Y	N
GQM/MEDEA [14]	Y	≥O	Y	N	Y

Table 1: Evaluation of formal software measurement approaches (Y= Yes, N = No, \ = Not applicable, O = Ordinal, I = Interval, R = Ratio, All = All types of scale)

Basically, the distance-based approach to software measurement models software attributes as distances. Metrics or distance functions are then considered as 'measures' of distance. They can thus be used to define measures for the software attributes that are modelled as distances.

In section 4 we use a measurement structure from measurement theory to define the concept of distance, independent from its measurement. We also present the conditions under which a metric is really a measure of distance, i.e., a measure in the sense of measurement theory. We further show that the metrics defined by the distance-based approach satisfy these conditions. In the present section we focus upon the process model and the constructive attribute modelling and measure definition procedures of the DISTANCE framework, without regard to the underlying measurement theoretic foundation.

3.1 Overview of DISTANCE

Before detailing the individual tasks of the attribute modelling and measure definition procedures, we provide an overview of DISTANCE using a process model (Fig. 1). The figure uses the notation of the UML activity diagram as this is an excellent instrument to model a workflow or procedure. For the precise semantics of UML activity diagrams we refer to [5].

The distance-based measure construction process consists of five steps to be executed in the order prescribed by the activity diagram.[5] The individual steps are further discussed in sub-sections 3.2 to 3.6.

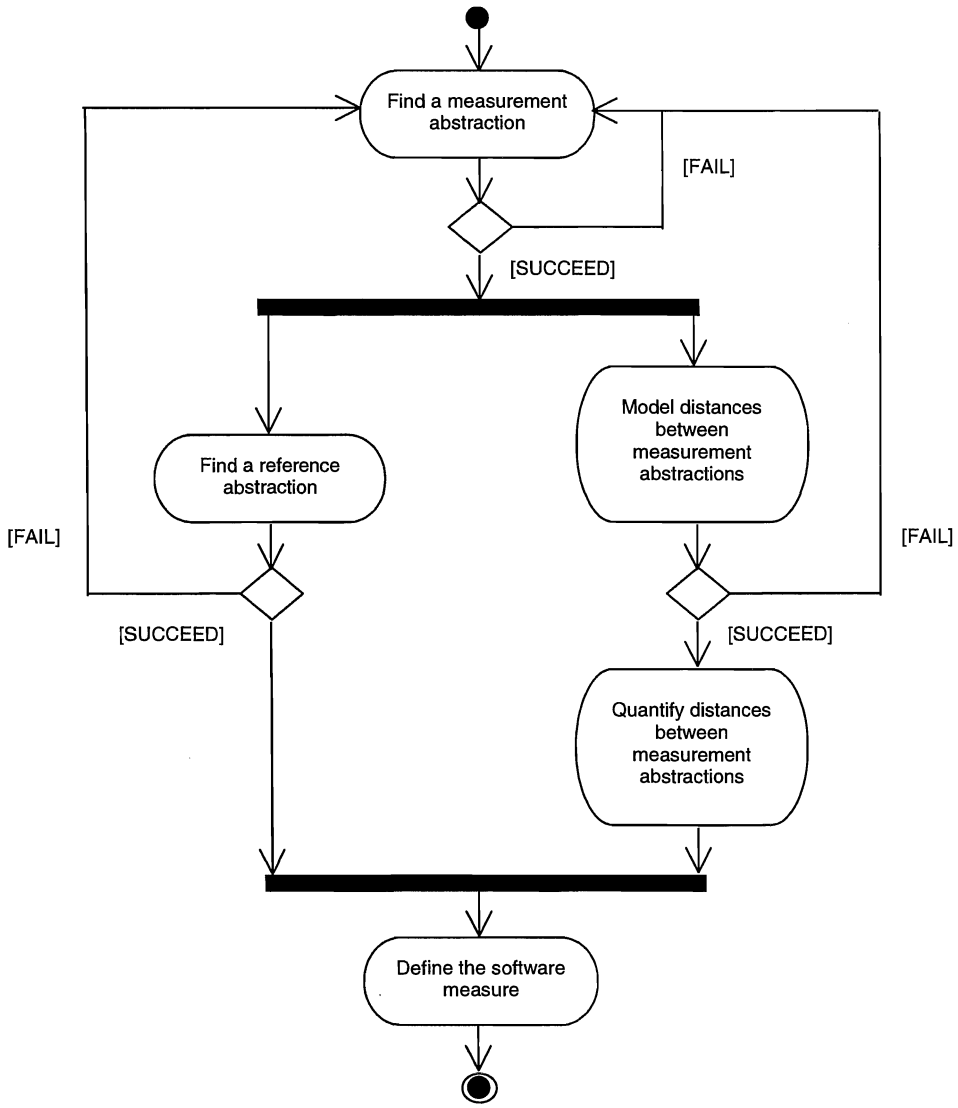

Figure 1: Process model for distance-based software measurement

[5] These steps are modelled as activity states in the activity diagram using lozenge shapes (i.e., symbols with horizontal top and bottom and convex sides). Transitions between the activity states, modelled as directed lines, indicate the transfer of control between activities.

The process is triggered by a request to find or build a measurement instrument for a software attribute *attr* that characterises the software entities in a set P. To illustrate distance-based software measurement throughout this section we use an example request that expresses the need to *measure the functionality of business objects*. We consider here business objects in the context of enterprise or business modelling, i.e., a business object is an object-oriented representation of a concept found in a business environment (e.g., a customer, a product, an order, etc.). Business objects encapsulate the static structure and dynamic behaviour of such concepts.

First, a measurement abstraction for the software entities must be found. The software entities of interest (e.g., business objects) must be modelled such that the attribute of interest (e.g., functionality) is emphasised. This means that the model should allow observing to what extent a software entity is characterised by the attribute. The result of the first step is a set of software entities M that can be used as measurement abstractions or models of the software entities in P for the attribute of interest *attr*. We also require a function *abs*: $P \rightarrow M$ that formally describes the rules of the mapping.

If the first step succeeds then two tracks of activities can be executed in parallel.[6] In the right-hand track of Fig. 1 the set M is defined as a metric space. First, distances between the elements of M are modelled as sequences of elementary transformations. Such a sequence represents a series of atomic changes applied to an element of M to arrive at another element of M. The number of atomic changes that are required to transform one element into the other determines the distance between these elements. The formal outcome of this step is a set T_e of elementary transformation types on M that must be used to build the sequences. Next, a metric δ: $M \times M \rightarrow \Re$ is defined to quantify the distances between the elements of M.

In the left-hand track of Fig. 1 a single step occurs. After having determined the measurement abstractions, we now need to determine what the model of a software entity in P must look like in case that entity would be characterised by the theoretical lowest value of *attr*. This hypothetical 'null' model or reference model can then be used as a reference point or norm for measurement. The result of this step is thus the definition of a function *ref*: $P \rightarrow M$ that returns for each software entity in P a reference abstraction for *attr* in M.

After having executed the two concurrent tracks, there is one final step in the distance-based measurement process.[7] It is this last step that expresses the basic idea of our approach. The software attribute *attr* is defined and measured as a specific distance within the metric space M. The extent to which *attr* characterises a software entity $p \in P$ is defined by the distance between the actual model of p for *attr* (i.e., *abs*(p)) and the reference model for *attr* (i.e., *ref*(p)). The larger this distance, the more the actual measurement abstraction differs from the norm that has been set and thus the greater the extent to which *attr* characterises p. Hence, the value of *attr* for p is the value returned by the metric δ for the pair (*abs*(p), *ref*(p)). The formal outcome of the last step is the measure μ: $P \rightarrow \Re$ defined such that $\forall\, p \in P$: $\mu(p) = \delta(abs(p), ref(p))$. For our example, the execution of the last step would result in the definition of a measure for the functionality of business objects. The construction of such a functionality measure and the distance-based definition of *business object functionality* will be further

[6] This is indicated in the activity diagram by a fork symbol (i.e., a horizontal black bar with one incoming transition and two or more outgoing transitions).

[7] In the activity diagram a join symbol (i.e., a horizontal black bar with one outgoing transition and two or more incoming transitions) is used to synchronise concurrent flows of control.

illustrated in the following subsections. Table 2 summarises the required inputs and expected results of the various steps in Fig. 1.

STEP	INPUTS	OUTPUTS
Find a measurement abstraction	The attribute of interest *attr* A set of software entities P	A set of software entities M (to be used as measurement abstractions) A function *abs*: $P \rightarrow M$
Model distances between measurement abstractions	M	A set of elementary transformation types T_e
Quantify distances between measurement abstractions	M T_e	A metric δ: $M \times M \rightarrow \Re$
Find a reference abstraction	*attr* P M	A function *ref*: $P \rightarrow M$ (to return a reference abstraction for *attr*)
Define the software measure	P *abs* δ *ref*	A function μ: $P \rightarrow \Re$

Table 2: Required inputs and expected results

We have not yet commented upon the branches found in Fig. 1.[8] These refer to a number of technical assumptions underlying the approach. If such an assumption does not hold in reality, then the process cannot be further executed, and some or more of the previous steps need to be executed again — that is why these assumptions are called 'technical'. The nature and impact of each of these assumptions is discussed in the remaining subsections.

Before discussing the individual tasks in the distance-based measure construction process, we need to stress one further aspect of DISTANCE. The process model in Fig. 1 gives the impression of describing the workflow of an independent measure construction process. However, by itself this process cannot function as it relies upon information provided from outside. In the context of empirical software engineering, the distance-based measurement process can be considered as the measure construction component of a larger process describing the flow of work in a typical empirical study. Finding measures for the variables investigated in such a study is only one activity, albeit an important one given the effect that construct validity has on the overall validity of the empirical study.

The contact points with the rest of the empirical study process are represented by the initial and final states of the process model in Fig. 1. The list of inputs and outputs in Table 2 helps identifying the preconditions of the workflow's initial state and the postconditions of the workflow's final state. The initial state is entered when a request for a measure is issued. This request stipulates the attribute of interest and the set of software entities characterised by the attribute. These are the inputs required for the execution of the first step in the process. The final state is reached after successful completion of the last step in the process. The request is answered by providing a measure definition. The further outputs found in Table 2 help to document the measure construction process such that the construct validity of the measure can be investigated. They also help to interpret the results of the empirical study. In fact, the formal attribute definition is as important as the measure definition itself.

[8] A branch symbol in an activity diagram is modelled as a diamond having one incoming transition and two or more outgoing transitions.

Fig. 2 is an example of how the distance-based measure construction process can be incorporated as a component into an empirical study process. The example takes the perspective of goal-oriented measurement, i.e., the development of measures and the empirical study is seen in the wider context of the corporate strategy and its long-term and short-term objectives (e.g., a software process improvement program).

The study goals are derived from the corporate goals by means of a goal setting technique. A popular template for such a technique has been provided by the Goal-Question-Metric (GQM) Paradigm of Basili *et al.* [4].[9] The template requires a study goal to be specified along four dimensions [3]:

- Issue (e.g., cost)
- Object of study (e.g., information system development projects)
- Viewpoint (e.g., project leader)
- Purpose (e.g., prediction)

This specific study goal might be derived from a corporate objective like *improve the accurateness of the budgeting process for information system development.*

Next, the study goal is translated into one or more empirical hypotheses [14]. For the example given we might for instance hypothesise that *the higher the functionality of a business object, the more it will cost to design, implement, and test its class definition.* Clearly, if we could accept this hypothesis, then we can examine the precise relationship between the functionality of a business model (i.e., a conceptual model of the business unit or process for which an information system must be built) and the cost of transforming that business model into a working information system. After modelling this relationship, a predictive model for information system development cost can be built, which helps to achieve the corporate goal.[10]

Apart from the identification of the empirical hypotheses, the empirical study must be designed. This requires that a number of choices must be made, for instance regarding the form of the study (e.g., an observational study, a controlled experiment, etc.), the sample size, the analysis techniques, etc.

After the set up of the empirical study the requests for measures of the independent and dependent variables in the empirical hypotheses can be issued. In our example we need a measure for the functionality of business objects (i.e., the independent variable) and a measure for the development cost of a business object class definition (i.e., the dependent variable). The example further focuses on the construction of a business object functionality measure.

---

[9] The definition of the study goals based on corporate goals is only one aspect of the GQM paradigm. Once the goals are determined, a set of questions is used to further refine the goals. These questions help identify the data that are needed for the empirical study. Although GQM is also intended as a measure construction or selection instrument, it does not specify how to develop or select the necessary measures to carry out the empirical study. Briand *et al.* [12], [14] for instance argue that GQM does not specify how to generate models of the object of study that capture the attribute(s) of interest (cf. the first step of the distance-based process). Their GQM/MEDEA approach (MEDEA stands for MEtric DEfinition Approach) extends GQM to remedy this problem (see also our discussion of GQM/MEDEA as a property-based approach in section 2).

[10] Of course, other hypotheses related to the study goal need to be tested as well, but these are omitted here for the sake of brevity.

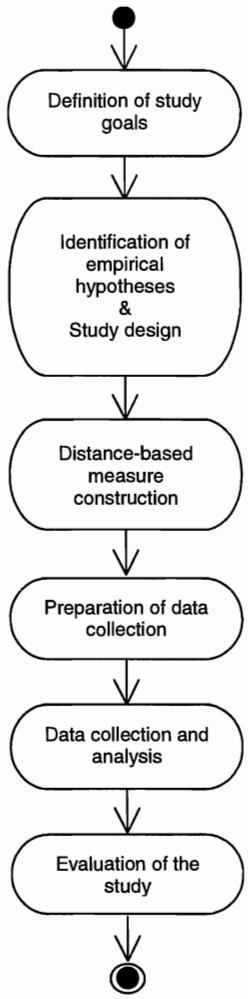

Figure 2: Distance-based measure construction as a component of the empirical study process

When measures have been developed, we need to consider issues of data collection (e.g., do we need a tool to automatically collect the measurement values?).[11] After collecting and analysing

[11] Although DISTANCE is presented as a measure construction framework, it can also be used for the purpose of measure selection. The *Preparation of data collection* activity then includes a comparison of the metric-based defined measure with existing measures. If a match is found between the measure function descriptions (i.e., domain, range, and mapping rule), then the existing measure can be used to measure the attribute of interest. In that case, the constructive procedures in DISTANCE did not result in the definition of a new measure, but in the formalisation and (construct) validation of an existing measure. A comparison with existing measures is particularly interesting in case the need for a CAME (Computer

the data, the empirical evidence is used to accept or refute the study hypotheses. Finally, the threats to the study validity must be evaluated, conclusions must be drawn, and the necessary actions must be taken (e.g., build a cost prediction model, reformulate the empirical hypotheses, replicate experiments, etc.). It must be noted that a realistic empirical study process is not as sequential as the one shown in Fig. 2. At some point in the process it might be needed to reconsider some or all of the previous steps. The iterative character of the process has been omitted from the model, as our only intention was to show how the distance-based measurement process fits into the global picture of empirical software engineering research. For a more detailed description of empirical software engineering studies we refer to [10], [12], [14].

3.2 Find a measurement abstraction

> For the set of software entities P that are characterised by an attribute *attr*, choose a set of software entities M that can be used as measurement abstractions to emphasise *attr*, and define a mapping *abs*: $P \rightarrow M$.

Measurement abstractions are used in software measurement to emphasise the attribute of interest, while simultaneously de-emphasising other attributes [59]. For instance, if we wish to measure to size of an object class in terms of the numerousness of its variables, then the state vector of the class is a suitable measurement abstraction. However, the state vector is not a suitable abstraction to capture the complexity of the object class methods.

Whatever attribute needs to be measured, a software entity must be represented in one way or another. This representation of a software entity (e.g., an object class) is itself a software entity (e.g., a state vector). Now, a good model of a software entity $p \in P$ is one that reflects the extent to which *attr* characterises p. This means that if p would be characterised differently by *attr* (or would have less of *attr*) than it actually is (or than it actually has), then p would be mapped into another element of M than its current measurement abstraction *abs*(p) $\in$ M. If such differences cannot be observed, then a wrong choice has been made. Hence, the technical assumption underlying this step is that a suitable measurement abstraction for *attr* can be found and that the mapping of the elements of P into their models in M can be described in an unambiguous manner.

It must be noted that the function *abs* is total, but neither a surjection, nor an injection (Fig. 3). The implications are respectively that each relevant abstraction of a software entity in P is contained in M, that not all elements of M are measurement abstractions of the software entities in P, and that different software entities in P may map into the same element of M.

In our example P is a set of business objects relevant for some universe of discourse (e.g., a financial institution) and modelled using some formalism. For simplicity's sake we assume here a high-level (or analysis) point of view and model a business object as a tuple $p = (\sigma, \alpha)$, where σ is the state vector of p, containing the attributes of p, and α is the alphabet of P, containing the business events that p is involved in.[12] At lower levels of abstraction, business objects are

---

Aided MEasurement) tool is felt. If such a tool is available for a matching measure, then we need not to build one ourselves.

[12] This formalism for business or enterprise modelling is related to the one found in [51]. Note that we describe business objects and business events as if they were business object *types* and business event *types*. The distinction between the type-level (e.g., customer), and the occurrence-level (e.g., customer 'John Peters') is not relevant here.

described using object class definitions in which attributes are given a data type declaration and business events correspond to methods. Some example, high-level, business object definitions are shown below.

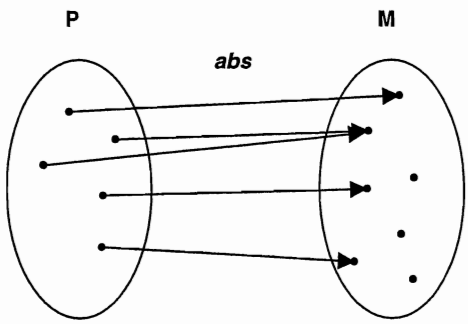

Figure 3: The abstraction function *abs*: P → M

ACCOUNT
State Vector (σ) Account_ID Type Balance Owner ...
Alphabet (α) Open_account Close_account Deposit Withdraw Add_interest Subtract_cost ...

CUSTOMER
State Vector (σ) Customer_ID Name Address Age Accounts Loans ...
Alphabet (α) Open_account Close_account Deposit Withdraw Get_loan Pay_debt Close_loan Change_address ...

The proper execution of the first step in the distance-based measurement process requires a basic understanding of the attribute of interest. Ideally, the attribute has been informally described in the request for measurement. Otherwise, some perspective on the attribute must be taken now. In either case it must be checked whether the notion of the attribute developed corresponds to the understanding of the attribute as expressed in the hypothesis of the empirical study.

For our example, we assume that the functionality of a business object is determined by its participation in business events. From a high-level, analysis point of view, the numerousness of the business events in the alphabet of the business object determines its functionality.[13] Hence, the measurement abstraction for a business object is its alphabet. Let U be the universe of business events associated with the universe of discourse. The alphabet of a business object is then a subset of U. The set that contains the alphabets of business objects as its elements is the power set of U, denoted by $\wp(U)$. The abstraction function *abs* is thus a projection function *abs*: $P \rightarrow \wp(U)$: $(\sigma, \alpha) \rightarrow \alpha$ that maps business objects onto their alphabet.

3.3 Model distances between measurement abstractions

Define a set T_e of elementary transformation types for M that is constructively complete and inverse constructively complete.

The next step is to model distances between the elements of M. To this end we use the concept of an *elementary transformation*. An elementary transformation of an element of M results in another element of M that is a modification of the first element. This modification is assumed to be atomic in the sense that it cannot be subdivided in a series of more elementary changes. Intuitively, the number of elementary transformations required to transform one element of M into another element represents the distance between these elements.[14] When used in this sense, elementary transformations are similar to the elementary edit operations for trees, such as defined for instance by Zhang *et al.* [57] and Oommen *et al.* [46]. Sequences of tree edit operations are used to define the edit distance between trees (called the tree-editing problem). Similar edit operations have been used to define distances between strings (i.e., the string-editing problem). In our approach, trees and strings could well be the elements of M. So, it makes sense to define distances in M in terms of elementary transformations between the elements of M.

Elementary transformations belong to *elementary transformation types*. These are homogenous functions on M that precisely describe the modification rule that must be applied to the argument of the function. We require that the set T_e of elementary transformation types for M is constructively complete, meaning that every element of M can be 'reached' by applying a finite sequence of elementary transformations on some initial (or base or 'null') element of M. We also require T_e to be inverse constructively complete, meaning that the initial element of M is 'reachable' from any other element in M by applying an appropriate finite sequence of elementary transformations. If T_e is both constructively and inverse constructively complete, then any element of M can be transformed into any other element of M, directly or via the initial element. This allows modelling the distance between any pair of elements in M.

[13] Again, for simplicity's sake, we do not wish to develop a complex notion of functionality here.

[14] This distance may be purely conceptual, for instance, the conceptual distance between two consecutive versions of a software package.

These ideas are formalised below. Each type of elementary transformation of a software product abstraction in M is defined through a homogenous function.

DEFINITION 2 (Elementary Transformation Type)

> An elementary transformation type for M is a function t_i: $M \rightarrow M$ mapping each element of M to an element of M according to prescribed mapping rules.

We require the set of elementary transformation types to be constructively complete.

DEFINITION 3 (Constructive Completeness)

> Let $T_e = \{t_0, t_1, \ldots, t_n\}$, where t_i, $i \in \{0, 1, \ldots, n\}$, is an elementary transformation type for M. Then, T_e is constructively complete if each $m \in M$ can be built by applying a finite sequence of elementary transformations $t_{i1}, \ldots, t_{ik}$, with $i \in \{0, 1, \ldots, n\}$, on an initial abstraction $m_I \in M$.[15]

We also require that T_e is inverse constructively complete.

DEFINITION 4 (Inverse Constructive Completeness)

> Let $T_e = \{t_0, t_1, \ldots, t_n\}$, where t_i, $i \in \{0, 1, \ldots, n\}$, is an elementary transformation type for M. Then, T_e is inverse constructively complete if $\forall\, t_i \in T_e$: $\exists!\, t_i^{-1} = t_j \in T_e$: $t_j(t_i(m)) = m$, where $m \in M$.

The technical assumption underlying this step is that a set of elementary transformation types can be found that satisfies the requirements of constructive completeness and inverse constructive completeness. In [47] it was shown that such a set of elementary transformation types exists for common structures as sets, multi-sets, matrices and regular expressions. Comparable sets for strings and trees can be found in the literature. So, from a pragmatic point of view the assumption will mostly hold in reality. Note however that only as recently as 1989 a satisfactory solution for the tree-editing problem was found [46]. The process model (Fig. 1) suggests choosing another type of measurement abstraction for the attribute of interest when no constructively complete and inverse constructively complete set of elementary transformation types for the original set of measurement abstractions can be found.

In the next step, T_e is used to define a metric space (M, δ). Let us first introduce the following definitions and notations.

DEFINITION 5 (Sequence of Elementary Transformations, T-derivation)

> (1) Let T be a sequence of elementary transformations $t_{i1}, \ldots, t_{ik}$. A T-derivation from $m \in M$ to $m' \in M$ is a sequence of abstractions $m_0, m_1, \ldots, m_{k-1}, m_k$ such that $m = m_0$, $m' = m_k$ and $t_{ij}(m_{j-1}) = m_j$ for $1 \leq j \leq k$.
>
> (2) We say that a sequence T of elementary transformations $t_{i1}, \ldots, t_{ik}$ takes $m \in M$ to $m' \in M$ if it defines a T-derivation from $m \in M$ to $m' \in M$. If T takes m to m' then we denote T as $T_{m,m'}$.

[15] Each elementary transformation t_{ij}, $j \in \{1, 2, \ldots, k\}$ in such a sequence is of a type $t_i \in T_e$, but it is not required that all elementary transformations t_{ij} are of the same type $t_i \in T_e$ for all $j \in \{1, 2, \ldots, k\}$.

(3) The length of a sequence T of elementary transformations $t_{i1}, \ldots, t_{ik}$ is k. If k = 0 then T is empty. An empty sequence is denoted by the symbol ∅.

(4) Generally, there are many different sequences to take m to m'. The set of all sequences of elementary transformations that take m ∈ M to m' ∈ M is denoted by $\boldsymbol{T}_{m,m'}$.

(5) The set of all shortest sequences of elementary transformations that take m ∈ M to m' ∈ M is $\boldsymbol{ST}_{m,m'} \subseteq \boldsymbol{T}_{m,m'}$.

(6) The sets of all sequences and shortest sequences of elementary transformations that take any m ∈ M to any m' ∈ M are $\boldsymbol{T}$ and $\boldsymbol{ST}$ respectively.

To illustrate the above definitions and notations we return to the example on business object functionality measurement. We need to find a constructively complete and inverse constructively complete set of elementary transformation types for the set of measurement abstractions $\wp(U)$. This is quite easy in case of a power set. Since the elements of $\wp(U)$ are sets of business events, T_e must only contain two types of elementary transformations: one for adding a business event to a set and one for removing a business event from a set. Given two sets of business events $\alpha_1 \in \wp(U)$ and $\alpha_2 \in \wp(U)$, α_1 can always be transformed into α_2 by removing first all business events from α_1 that are not in α_2, and then adding all business events to α_1 that are in α_2, but not in the original α_1. In the 'worst case scenario', α_1 must be transformed into α_2 via the empty set (i.e., the initial element of $\wp(U)$). Hence, T_e satisfies the requirements of constructive completeness and inverse constructive completeness. Formally, $T_e = \{t_0, t_1\}$, where t_0 and t_1 are defined as:
$t_0: \wp(U) \rightarrow \wp(U): \alpha \rightarrow \alpha \cup \{e\}$, with $e \in U$
$t_1: \wp(U) \rightarrow \wp(U): \alpha \rightarrow \alpha - \{e\}$, with $e \in U$

The sequence of elementary transformations $T_{abs(\text{CUSTOMER}),abs(\text{ACCOUNT})} = t_{01}, t_{02}, t_{13}, t_{14}, t_{15}, t_{16}$ can be used to define the following T-derivation from *abs*(CUSTOMER) to *abs*(ACCOUNT):
m_0 = {Open_account, Close_account, Deposit, Withdraw, Get_Loan, Pay_Debt, Close_Loan, Change_address}
m_1 = {Open_account, Close_account, Deposit, Withdraw, Get_Loan, Pay_Debt, Close_Loan, Change_address, Add_interest} = $t_0(m_0)$
m_2 = {Open_account, Close_account, Deposit, Withdraw, Get_Loan, Pay_Debt, Close_Loan, Change_address, Add_interest, Subtract_cost} = $t_0(m_1)$
m_3 = {Open_account, Close_account, Deposit, Withdraw, Pay_Debt, Close_Loan, Change_address, Add_interest, Subtract_cost} = $t_1(m_2)$
m_4 = {Open_account, Close_account, Deposit, Withdraw, Close_Loan, Change_address, Add_interest, Subtract_cost} = $t_1(m_3)$
m_5 = {Open_account, Close_account, Deposit, Withdraw, Change_address, Add_interest, Subtract_cost} = $t_1(m_4)$
m_6 = {Open_account, Close_account, Deposit, Withdraw, Add_interest, Subtract_cost} = $t_1(m_5)$

The length of $T_{abs(\text{CUSTOMER}),abs(\text{ACCOUNT})}$ is 6. Of course, $T_{abs(\text{CUSTOMER}),abs(\text{ACCOUNT})}$ is only one element of $\boldsymbol{T}_{abs(CUSTOMER),abs(ACCOUNT)}$. Other sequences of elementary transformations might do the job. However, it is obvious that $T_{abs(\text{CUSTOMER}),abs(\text{ACCOUNT})}$ is a shortest sequence of elementary transformations taking CUSTOMER to ACCOUNT (i.e., $T_{abs(\text{CUSTOMER}),abs(\text{ACCOUNT})} \in \boldsymbol{ST}_{abs(CUSTOMER),abs(ACCOUNT)}$).

3.4 Quantify distances between measurement abstractions

Using the procedure below, define a metric δ: $M \times M \rightarrow \Re$ such that (M, δ) is a metric space.

In this step the distances in M that were defined using T_e are quantified. This is done by means of a metric space. Use the following procedure:

a) Let c be a positive real number. Define a function φ as follows:
   φ: $\boldsymbol{T} \rightarrow \Re$: $T_{m,m'} \rightarrow$ k.c, where k is the length of $T_{m,m'}$
b) Define the metric δ as follows:
   δ: $M \times M \rightarrow \Re$: $(m, m') \rightarrow \varphi(T_{m,m'})$, where $T_{m,m'} \in \boldsymbol{ST_{m,m'}}$

The function φ returns for a sequence of elementary transformations the product of the length of that sequence (i.e., k) and a positive real number (i.e., c).[16] The function δ returns for each pair of elements (m, m') of M the value that is returned by φ for a shortest sequence of elementary transformations taking m to m'. Hence, *the distance between m and m' is measured by counting the elementary transformations in the shortest sequences of elementary transformations taking m to m', and multiplying this count by a positive constant.* In appendix B it is proven that a function δ defined with this procedure satisfies the metric axioms.

For the illustration case we need to define a metric space $(\wp(U), \delta)$. Applying the procedure to construct a metric results in:

a) Let c be a positive real number. Define a function φ as follows:
   φ: $\boldsymbol{T} \rightarrow \Re$: $T_{\alpha,\alpha'} \rightarrow$ k.c, where k is the length of $T_{\alpha,\alpha'}$
b) Define the metric δ as follows:
   δ: $\wp(U) \times \wp(U) \rightarrow \Re$: $(\alpha, \alpha') \rightarrow \varphi(T_{\alpha,\alpha'})$, where $T_{\alpha,\alpha'} \in \boldsymbol{ST_{\alpha,\alpha'}}$

The definition of δ: $\wp(U) \times \wp(U) \rightarrow \Re$ can be reformulated. Assume we have to transform $\alpha \in \wp(U)$ into $\alpha' \in \wp(U)$. As exactly one elementary transformation (t_0 or t_1) is needed for each business event of U that is contained in either α or α', but not in both sets, the length of a shortest sequence of elementary transformations taking α to α' is equal to the cardinality of the symmetric difference between α and α'.[17] Hence, the above definition of δ may be reformulated as:

Let c be a positive real number. Define a metric δ as follows:
δ: $\wp(U) \times \wp(U) \rightarrow \Re$: $(\alpha, \alpha') \rightarrow c.(|\alpha - \alpha'| + |\alpha' - \alpha|)$

For c = 1 this metric takes the form of the symmetric difference model, which is a special case of Tversky's contrast model that is used to quantify the dissimilarity between two sets of features [52]. The symmetric difference model results in a value of 6 for the distance between the alphabets of the CUSTOMER and ACCOUNT business objects:

δ(*abs*(CUSTOMER),*abs*(ACCOUNT))
= |{Open_account, Close_account, Deposit, Withdraw, Get_Loan, Pay_Debt, Close_Loan, Change_address} - {Open_account, Close_account, Deposit, Withdraw, Add_interest, Subtract_cost}| + |{Open_account, Close_account, Deposit, Withdraw, Add_interest,

[16] Within a metric space (M, δ), the positive real number c is a constant. For the sake of convenience, it is often chosen to be equal to one. However, other values can be chosen for the purpose of 'rescaling' (cf. section 4).

[17] The symmetric difference between α and α' (notation: $\alpha \Delta \alpha'$) is the union of $\alpha - \alpha'$ and $\alpha' - \alpha$.

Subtract_cost} - {Open_account, Close_account, Deposit, Withdraw, Get_Loan, Pay_Debt, Close_Loan, Change_address} |
= |{Get_Loan, Pay_Debt, Close_Loan, Change_address}| + |{Add_interest, Subtract_cost}|
= 6

3.5 Find a reference abstraction

> Define a function *ref*: P → M such that, for all p ∈ P, *ref*(p) = *abs*(p) if and only if p has the theoretical lowest value of *attr*.

This is a crucial step in the distance-based measure construction process as it reflects our understanding of the attribute being measured. We need to carry out a sort of thought experiment. We must imagine what *abs*(p) would look like if p is characterised by the theoretical lowest value of the attribute. This imaginary representation of p is called the reference abstraction of p for *attr*. It can be argued that the closer *abs*(p) to this reference abstraction, the lower the value of the attribute. Analogously, the farther *abs*(p) from the reference abstraction, the higher the value of the attribute. Hence, the (conceptual) distance between the actual measurement abstraction and the reference abstraction is used as a model of the attribute of interest. For each entity in P it is this particular distance that needs to be measured.

Formally, the reference abstraction is returned by a function *ref*: P → M. As *ref* is a function there is exactly one reference abstraction associated with an entity p ∈ P for the attribute of interest *attr*. The assumption that this reference abstraction can be identified is essential to DISTANCE. We believe however that this (technical) assumption is mostly fulfilled in reality. We only require that, prior to measurement, we have an idea of the theoretical lowest value of the attribute and we can identify the software entities in P that are characterised by the lowest attribute value.[18] The measurement abstraction returned by *abs* for these entities helps to find the mapping rule for *ref*. According to our experiences with object-oriented software measurement [47] (cf. also section 5), it is common that *ref* maps all elements of P into a same reference abstraction, which is often the initial element of M. If no function *ref* can be defined (e.g., for a given software entity p, the reference model for *attr* cannot be uniquely determined), then the process model (Fig. 1) suggests to use another measurement abstraction (and thus another function *abs*) to emphasise the attribute of interest.

For the example on business object functionality measurement, the obvious reference point for measurement is the empty alphabet. It can be argued that a business object that is not involved in any business event has no functionality whatsoever. Perhaps such business objects do simply not exist in reality, but from a theoretical point of view, they would be characterised by the lowest value of functionality. Every business object that participates in at least one business event has a higher functionality. Hence, we define the following function:
ref: P → $\wp$(U): p → ∅

[18] The distance-based approach cannot be used for attributes that are only nominal classifications of software entities. There is no ranking between the equivalence classes in a nominal classification. Hence, there is no theoretical lowest value.

3.6 Define the software measure

Define a function μ: P $\rightarrow \Re$ such that, for all p $\in$ P, μ(p) = δ(*abs*(p), *ref*(p)).

The results of the previous activities are now combined to produce a formal definition of the attribute of interest *attr* and its measure. Formally, *attr* of p $\in$ P is the distance between *abs*(p) and *ref*(p) as modelled by the shortest sequences of elementary transformations taking *abs*(p) to *ref*(p) (i.e., as modelled by $T_{abs(p),ref(p)} \in \boldsymbol{ST}_{abs(p),ref(p)}$). The value for this distance is δ(*abs*(p), *ref*(p)). Hence, if μ: P $\rightarrow \Re$ is defined such that μ(p) = δ(*abs*(p), *ref*(p)), then μ can be used as a 'measure' of *attr*. In the next section it is shown that the function μ: P $\rightarrow \Re$ is really a measure (in the sense of measurement theory) of the software attribute *attr*.

To avoid any confusion that might arise, we explicitly state at this point that not every real value δ(m, m'), with m, m' $\in$ M, is a measurement of the software attribute of interest. For a set of software entities P characterised by the attribute *attr*, the set of measurements contains only the metric values δ(*abs*(p), *ref*(p)), where *abs*(p) $\in$ M is the measurement abstraction of p $\in$ P for *attr*, and *ref*(p) $\in$ M is the reference abstraction of p for *attr*. Distances involving other elements of M do not qualify as definitions of the software attribute *attr*. Similarly, the distance measurements involving such elements of M have nothing to do with the measurement of *attr*.

In our illustration case, the functionality of a business object p $\in$ P can now be defined as the distance between its alphabet *abs*(p) and the empty set *ref*(p), as modelled by $T_{abs(p),ref(p)} \in \boldsymbol{ST}_{abs(p),ref(p)}$.

Given that *ref*(p) = $\varnothing$ for all p $\in$ P, it holds that μ(p) = δ(*abs*(p), *ref*(p))
= $|$*abs*(p) - *ref*(p)$|$ + $|$*ref*(p) - *abs*(p)$|$
= $|$*abs*(p) - $\varnothing|$ + $|\varnothing$ - *abs*(p)$|$
= $|$*abs*(p)$|$

As a consequence, a measure that returns the count of business events in the alphabet of a business object qualifies as a business object functionality measure. The measurement values for CUSTOMER and ACCOUNT are 8 and 6 respectively.

4 MEASUREMENT THEORETIC EVALUATION

In this section we investigate DISTANCE from the perspective of measurement theory. In the introduction it was stated that the focus of this paper is the construct validity of the software measures used in empirical research. Apart from offering a set of procedures to model software attributes and define software measures, integrated into a process model for measure construction, a framework for measurement must be consistent with measurement theory. The framework must clarify how, and under what conditions, the construct validity of the measures is guaranteed. The framework must also clarify the type(s) of scale that is supported.

We first present a measurement structure from measurement theory that formalises the idea of 'distance measurement'. Next we relate the distance-based approach to this measurement structure.

4.1 Proximity measurement

To define a metric as a measure of distance it must be a homomorphism from an empirical relational system into a numerical relational system. The usual axiomatic definition of a metric (i.e., Def. 1) only considers numerical properties. To define a metric as a homomorphism, the empirical relational system must be defined as well. Measurement theory considers both qualitative and quantitative structures. The following observation by Suppes *et al.* ([52], p. 2) is illustrative: "There are, in fact, two quite distinct developments to be considered: analytic geometry, which formulates the spaces of numerical geometrical structures that potentially may serve to represent qualitative geometrical structures, and synthetic geometry, which develops the axiomatic theories of those qualitative structures. The pattern is the same as in measurement theory: a representation theorem shows how to embed a qualitative structure isomorphically into some family of numerical structures, and the corresponding uniqueness theorem describes the different ways that the embedding is possible".

Measurement theory describes the concept of distance or dissimilarity (i.e., a conceptual distance) by means of an empirical relational structure called a proximity structure.[19]

DEFINITION 6 (Proximity Structure)

(1) Let A be a set. The quaternary relation $\bullet\geq$ on A (i.e., $\bullet\geq \subseteq A^4$) is a binary relation on $A \times A$ (i.e., $\bullet\geq \subseteq A^2 \times A^2$) such that $\forall\, a, b, c, d \in A$: $(a, b) \bullet\geq (c, d)$ if and only if the dissimilarity between a and b is at least as great as that between c and d.

(2) The quaternary relation $\approx$ on A is an equivalence relation on $A \times A$ such that $\forall\, a, b, c, d \in A$: $(a, b) \approx (c, d) \Leftrightarrow (a, b) \bullet\geq (c, d)$ and $(c, d) \bullet\geq (a, b)$.

(3) The quaternary relation $\bullet>$ on A is a strict ordering relation on $A \times A$ such that $\forall\, a, b, c, d \in A$: $(a, b) \bullet> (c, d) \Leftrightarrow (a, b) \bullet\geq (c, d)$ and not $((c, d) \bullet\geq (a, b))$.

(4) $(A, \bullet\geq)$ is a proximity structure if and only if the following axioms hold $\forall\, a, b \in A$:

P_1. $\bullet\geq$ is a weak order;
P_2. $(a, b) \bullet> (a, a)$ whenever $a \neq b$ *(positivity)*;
P_3. $(a, a) \approx (b, b)$ *(minimality)*;
P_4. $(a, b) \approx (b, a)$ *(symmetry)*.

Part (1) of Def. 6 states that dissimilarity (or distance) is an attribute of a pair of elements. Prior to measurement, there exists an order $\bullet\geq$ on the dissimilarities between the pairs of elements of some set A. Parts (2) and (3) derive an equivalence relation $\approx$ and a strict ordering relation $\bullet>$ from the dissimilarity ordering $\bullet\geq$. Finally, part (4) describes the empirical conditions that must hold for the pair $(A, \bullet\geq)$ to qualify as a proximity structure. Condition P_1 requires the empirical dissimilarity ordering $\bullet\geq$ to be a weak order, i.e., it must be transitive and strongly complete. Conditions P_2, P_3 and P_4 follow from the metric axioms non-negativity, symmetry and identity. The fact that any metric δ satisfies $\delta(a, b) = \delta(b, a) > \delta(a, a) = \delta(b, b)$ for all $a \neq b$ requires the empirical conditions of positivity, minimality and symmetry.

The distance-based approach constructs a special kind of proximity structure, i.e., a segmentally additive proximity structure. The representation of such a structure has more favourable properties than the representation of a proximity structure.

[19] The definitions in this subsection are taken from Suppes *et al.* [52].

DEFINITION 7 (Collinear Betweenness, Segmentally Additive Proximity Structure)

(1) Let A be a set. For $a, b, c \in$ A, the ternary relation $\langle abc \rangle$ is said to hold if and only if $\forall\, a', b', c' \in$ A: $(a, b) \bullet\!\geq (a', b')$ and $(a', c') \bullet\!\geq (a, c) \Rightarrow (b', c') \bullet\!\geq (b, c)$

(2) A relational structure (A, $\bullet\!\geq$) is a segmentally additive proximity structure if and only if the following axioms hold for all $a, b, c, d \in$ A:
S_1. (A, $\bullet\!\geq$) is a proximity structure;
S_2. If $(a, b) \bullet\!\geq (c, d)$, then there exists $e \in$ A such that $(a, e) \approx (c, d)$ and $\langle aeb \rangle$;
S_3. If $c \neq d$, then there exist $e_0', \ldots, e_n' \in$ A such that $e_0' = a$, $e_n' = b$ and $(c, d) \bullet\!\geq (e_{i-1}', e_i')$, for all i = 1, ..., n.

The ternary relation $\langle abc \rangle$ is a collinear betweenness relation. Informally, $\langle abc \rangle$ holds if b lies on an additive segment from a to c, i.e., along the segment from a to c distances are additive. The extra axioms of the segmentally additive proximity structure (i.e., S_2 and S_3) are harder to interpret than those of the proximity structure (i.e., S_1). Axiom S_2 is the segmental solvability condition. It tells us that given two distances (a, b) and (c, d), the former being not less than the latter, we can always find an additive segment from a to e on (a, b) that is exactly equal to the distance between c and d. Axiom S_3 then guarantees that (c, d) cannot be infinitely small compared with (a, b). Informally, S_2 and S_3 were introduced to create a 'unit of distance'. These form the additive segments of any distance between elements of A, and thus allow a quantitative assessment of the relative proportion between distances.

A representation of a segmentally additive proximity structure is called a representation by a metric with additive segments. The following representation and uniqueness theorem can be found in Suppes *et al.* [52].

THEOREM 1

Let (A, $\bullet\!\geq$) be a segmentally additive proximity structure. Then there exist a real-valued function δ on A × A such that, for any $a, b, c, d \in$ A,
M_1. (A, δ) is a metric space;
M_2. $(a, b) \bullet\!\geq (c, d) \Leftrightarrow \delta(a, b) \geq \delta(c, d)$;
M_3. $\langle abc \rangle \Leftrightarrow \delta(a, b) + \delta(b, c) = \delta(a, c)$;
M_4. If δ' is another metric on A satisfying the above conditions, then there exist $\beta > 0$ such that $\delta' = \beta\delta$.

The term 'metric' is generally understood as referring to a 'measure' of distance. The theorem states the empirical conditions that the attribute of distance must satisfy in order to define a metric as a measure in the sense of measurement theory (i.e., as a homomorphism). These empirical conditions are those associated with the (segmentally additive) proximity structure.

If the metric δ would only satisfy M_1 and M_2, then (A, $\bullet\!\geq$) must only be a proximity structure (i.e., S_1 or P_1 to P_4 must be satisfied). The representation is then essentially an ordinal representation [52].

If in addition M_3 is satisfied, then (A, $\bullet\!\geq$) must be a segmentally additive proximity structure, implying that distance must also be characterised by S_2 and S_3. The metric δ is then called a metric with additive segments. The values returned by δ for the additive segments on a

distance from a to c must add up to the value that is returned for (a, c) itself. It should be noted that M_3 is similar to, but weaker than the additivity condition of an extensive representation.

Condition M_4 is actually a uniqueness theorem. It tells us that the scale type of a metric with additive segments is ratio (as the class of admissible transformations is that of the similarity transformations).

The representation by a metric with additive segments belongs to the class of the unidimensional spatial proximity representations.

4.2 Measure validation

We now relate the theorem of proximity measurement to the distance-based approach. The steps *Find a measurement abstraction* and *Model distances between measurement abstractions* result in the definition of a segmentally additive proximity structure. Together with step *Find a reference abstraction*, they can be considered as a constructive attribute modelling procedure. The execution of these steps proceeds prior to measurement and thus formalises the empirical understanding of the attribute of interest. The procedure to define a metric in step *Quantify distances between measurement abstractions* results in the definition of a metric space with additive segments. This step, together with step *Define a software measure*, forms the measure definition procedure. The process model of DISTANCE (Fig. 1) indicates that once the attribute has been successfully modelled, the construction of a measure function proceeds automatically. No further technical assumptions are required.

4.2.1 The empirical relational system

To define the distance between the elements of the set M in terms of a segmentally additive proximity structure $(M, \bullet\geq)$, we need to define the empirical dissimilarity ordering $\bullet\geq$. Let this order be determined by the following observation criterion: If m_1, m_2, m_3, and m_4 are elements of M, then $(m_1, m_2) \bullet\geq (m_3, m_4)$ is observed when the length of the sequences in $\boldsymbol{ST_{m1,m2}}$ is greater or equal than the length of the sequences in $\boldsymbol{ST_{m3,m4}}$. Put differently, prior to measurement, we state that the dissimilarity between m_1 and m_2 is greater or equal than the dissimilarity between m_3 and m_4 when we need at least as many elementary transformations to take m_1 to m_2 than to take m_3 to m_4. This observation criterion corresponds to our intuitive notions of distance in M such as motivated before (cf. sub-section 3.3).

In Appendix A we prove that if the above observation criterion is used to define the dissimilarity ordering on M, then the empirical conditions of the segmentally additive proximity structure are satisfied. Consequently, the execution of the steps *Find a measurement abstraction* and *Model distances between measurement abstractions* results in the definition of a segmentally additive proximity structure for the set M.

In the business object functionality example, the segmentally additive proximity structure $(\wp(U), \bullet\geq)$ is defined. The dissimilarity ordering $\bullet\geq$ is defined by the following observation criterion: for the sets of business events $\alpha_1, \alpha_2, \alpha_3, \alpha_4 \in \wp(U)$, $(\alpha_1, \alpha_2) \bullet\geq (\alpha_3, \alpha_4)$ is observed when we need at least as many elementary transformations that add a business event to α_1 or that delete a business event from α_1 to arrive at α_2 than we need elementary transformations (adding or deleting business events) to take α_3 to α_4.

4.2.2 The numerical relational system

Given the segmentally additive proximity structure (M, $\bullet\geq$), the representation theorem for proximity structures (Theorem 1) now guarantees that there exists a representation by a metric with additive segments. The function δ: M × M → $\Re$ defined in step *Quantify distances between measurement abstractions* is exactly such a metric. The verification of the numerical conditions is presented in Appendix B. Hence, δ is a measure of dissimilarity in the sense that it homomorphically preserves the empirical conditions of the segmentally additive proximity structure (M, $\bullet\geq$). Moreover, according to the uniqueness theorem it is characterised by the ratio scale type. Note that the admissible transformations for the ratio scale type are implicitly incorporated in the constructive procedure through the choice of a positive real number c. If the value of c is changed (e.g., for the purpose of rescaling the measurement values), then a new metric δ' is defined that also homomorphically preserves the empirical conditions of (M, $\bullet\geq$). Both δ and δ' define a ratio scale of dissimilarity.

In the business object functionality example, the metric space with additive segments ($\wp$(U), δ), with δ: $\wp$(U) × $\wp$(U) → $\Re$: (α, α') → c.($|\alpha - \alpha'| + |\alpha' - \alpha|$) (c > 0), is constructed. For c = 1, δ returns the cardinality of the symmetric difference between two sets of business events. According to Theorem 1, δ homomorphically preserves the empirical dissimilarity ordering $\bullet\geq$ on $\wp$(U), i.e., δ is a measure of dissimilarity (or distance) for sets of business events.

4.2.3 Construct validity of the software measures

We did not show yet that metric values can be used as valid measurements of the attribute of interest. Technically, it is not hard to prove that the distance-based approach defines a measure for the attribute of interest. Recall that, for each software entity p ∈ P, *attr* of p is the distance from *abs*(p) ∈ M to *ref*(p) ∈ M as defined by the sequences of elementary transformations $T_{abs(p),ref(p)} \in \boldsymbol{ST}_{abs(p),ref(p)}$. This distance is measured by $\mu(p) = \delta(abs(p), ref(p))$. Hence, μ: P → $\Re$ is a measure of *attr*.

However, the crucial assumption for the (construct) validity of μ is that the person that performs measurement agrees that *attr* of p is defined as the (conceptual) distance between *abs*(p) and *ref*(p). Basically, this means that the person performing measurement must agree that each elementary transformation in $T_{abs(p),ref(p)} \in \boldsymbol{ST}_{abs(p),ref(p)}$ contributes the same positive quantity to the *attr* of p. This implies that *attr* of p is proportional to the minimum number of elementary transformations needed to transform *abs*(p) into *ref*(p). Each of these elementary transformations represents the same positive amount of attribute, no matter what its position in a shortest sequence of elementary transformations is, or what type of change it causes. In that case, *attr* of p can truly be characterised by the dissimilarity or distance between *abs*(p) and *ref*(p). Moreover, the concept of elementary transformation may then be considered as a unit of distance or dissimilarity. Note that Kitchenham *et al.* [36] explicitly require the identification of equivalent measurement units in the context of software measure validation. Our approach is thus based on the same requirement.

In the example, functionality of a business object is distance-based defined as follows: *the functionality of a business object p is the distance between the alphabet of p and the empty set*. The metric-based definition of a business object functionality measure is: *the functionality of a business object p is measured by the count of business events in the alphabet of p*. As this

measure is characterised by the ratio scale type we may multiply all functionality values by a same positive real constant to get another business object functionality scale.[20]

5 CASE STUDY: A DISTANCE-BASED DEFINITION OF THE MOOSE MEASURES

Chidamber *et al.* [19], [20] have proposed a suite of six object-oriented design measures, called MOOSE (Metrics for Object Oriented Software Engineering). MOOSE is meant to measure 'internal' attributes of a class in an object-oriented design that are related to 'external' attributes (cf. Table 3).[21]

MOOSE MEASURE	INTERNAL ATTRIBUTE OF CLASS[22]	RELATED EXTERNAL ATTRIBUTES[23] (Chidamber *et al.* [20])
WMC - **W**eighted **M**ethods per **C**lass	Complexity	Development time and effort Maintenance time and effort Impact of changes and defects on children Reusability
DIT - **D**epth of **I**nheritance **T**ree	Depth in inheritance graph	Behavioural predictability Reusability
NOC - **N**umber **O**f **C**hildren	Numerousness of children	Reusability Likelihood of defects Testing effort
CBO - **C**oupling **B**etween **O**bject classes	Coupling	Reusability Sensitivity to changes Maintenance difficulty Testing effort
RFC - **R**esponse **F**or a **C**lass	Response for a class	Testing difficulty Debugging difficulty
LCOM - **L**ack of **CO**hesion in **M**ethods	Lack of cohesion in methods	Likelihood of defects

Table 3: The MOOSE measure suite of Chidamber and Kemerer

[20] The example looks trivial. The business object functionality measure resulting from the distance-based measure construction process is simply a count of object features (i.e., business events in the alphabet). It must be noted however that other measurement theoretic approaches, based on extensive measurement or belief functions, cannot be used to show that a count of object features is a measure characterised by the ratio scale type (for a detailed discussion see [59]).

[21] The distinction between internal and external software attributes is due to Fenton [26]. An attribute of a software entity is called internal if it can be evaluated without regard to the environment in which the entity functions. External attributes cannot be evaluated by isolating the object of study from its environment. For instance, reliability is an external attribute as it relates to the (in)correct behaviour of the software entity within a particular set of working conditions. A piece of avionics software might be reliable when used in clear weather, but might fail to work in a thunderstorm. The size of this piece of avionics software is in both environments the same. Size is an internal attribute.

[22] Precise (distance-based) definitions of these attributes are presented further in the paper.

[23] The external attributes are presented without definition. We refer to [20] for more details on the precise meaning of the hypothesised relationships.

Numerous empirical studies have used the MOOSE measures in the process of corroborating (or falsifying) hypothesised relationships between internal and external attributes of object-oriented software [2], [7], [15], [18], [40], [41], [50]. Other studies have criticised the work of Chidamber *et al.* from different angles. Churcher *et al.* [22] and Briand *et al.* [9] point at gaps in the OO formalism (e.g., polymorphism, method overriding and information hiding) that result in ambiguous measure definitions. Henderson-Sellers [34] criticises the incorrect use of mathematical concepts and terminology in some of the definitions. Hitz *et al.* [35] question the measurement theoretical foundation of some of the measures. Zuse [58] shows that many of the measures do not fit into the framework of extensive measurement and consequently argues that their use to define ratio scales is questionable.

It must be acknowledged that Chidamber *et al.* are quite rigorous in their approach to define a measure suite. For instance, many object-oriented constructs (e.g., object) and concepts (e.g., coupling, complexity) are defined using ontological principles. However, this ontological basis is no substitute for a formal attribute definition in terms of an empirical relational structure. As a consequence it is unclear which measurement structure was used by Chidamber *et al.* and how the construct validity of the measures can be proven. Therefore we decided to redefine the MOOSE suite using the DISTANCE framework. A distance-based definition of the attribute and a metric-based definition of the measure ensure the construct validity of the measures (via the representation theorem of proximity measurement), which is essential for the ultimate validity of the empirical studies using these measures. At the same time we hope that this exercise helps demonstrating the usefulness of DISTANCE.[24]

In order to unambiguously refer to object-oriented constructs we need a formalism and a standardised terminology. In sub-section 5.1 we follow a proposal of Briand *et al.* [9] presenting implementation independent definitions of object-oriented systems and their components. This formalism has been used by Briand *et al.* to evaluate a number of object-oriented coupling measures. We only present the definitions that have actually been used in the MOOSE case study. We further added some definitions of our own where the need to do so was felt. For the complete list of definitions in the original proposal and for an explanation of common object-oriented principles and constructs we refer to [9]. In sub-section 5.2 the redefined MOOSE measure suite is presented and discussed.

5.1 OO terminology and formalism

All definitions are taken from [9], except for definitions 9, 10, 13 and 19.

The basic construct in an object-oriented system is the class, or more precisely, the class definition of a type of objects. Inheritance is a classification mechanism for the classes in an object-oriented system.

DEFINITION 8 (System, Classes, Inheritance Relationships)

An object-oriented system consists of a set of classes, C. There can exist inheritance relationships between classes such that for each class $c \in C$ let

- *Parents*(c) $\subset$ C be the set of parent classes of c;
- *Children*(c) $\subset$ C be the set of children classes of c;
- *Ancestors*(c) $\subset$ C be the set of ancestor classes of c;[25]

[24] A similar study has been done by Zuse in [58], [59] using a measurement structure based on belief functions. However, the belief function approach of Zuse does not support the ratio scale type, whereas proximity measurement does.

[25] A parent is a direct ancestor, i.e., a class directly inherits from a parent class.

- *Descendants*(c) $\subset$ C be the set of descendant classes of c.[26]

For notational convenience we need an expression to universally qualify classes. The 'universe of classes' is the set of classes across the boundaries of the individual object-oriented systems.

DEFINITION 9 (Universe of Classes)

The universe of classes is denoted by UC. Each object-oriented system consists of a subset of UC.

The inheritance graph of an object-oriented system must not be connected. Classes that are not involved in inheritance relationships form an unconnected component in the graph. Root classes do not inherit from other classes. Leaf classes are not inherited from. A level of the inheritance graph is a set of classes at a certain depth in the graph. The same class can be on different levels in the graph as a consequence of multiple inheritance.

DEFINITION 10 (Inheritance Graph)

The inheritance graph of an object-oriented system C contains the classes of C as vertices and the inheritance relationships between the classes of C as edges. Let
Root(C) $\subseteq$ C be the set of root classes of the inheritance graph;
Leaf(C) $\subseteq$ C be the set of leaf classes of the inheritance graph;
$Level_i(C) \subseteq C$, for $i \in \{0, 1, \ldots\}$, be the set of classes at level i in the inheritance graph, where $Level_i(C)$ is recursively defined:
$Level_0(C) = Root(C)$
$Level_{i+1}(C) = \{c \in C \mid \exists\, c' \in Level_i(C): c \in Children(c')\}$

A class has a set of methods. Some methods are only declared in the class. Others are implemented in the class. Expressions are needed to denote all methods in the system and the universe of methods across systems.

DEFINITION 11 (Declared and Implemented Methods)

For each class $c \in C$ let M(c) be the set of methods of c. $M(c) = M_D(c) \cup M_I(c)$ where

- $M_D(c)$ is the set of methods declared in c, i.e., methods that c inherits but does not override or virtual methods of c;
- $M_I(c)$ is the set of methods implemented in c, i.e., methods that c inherits but overrides or nonvirtual noninherited methods of c.

DEFINITION 12 (The Set of all Methods)

M(C) is the set of all methods in the system and is represented as $M(C) = \bigcup_{c \in C} M(c)$.

DEFINITION 13 (Universe of Methods)

The universe of methods is denoted by UM. For each object-oriented system C, M(C) is a subset of UM.

A method of a class is invoked when a message is sent to an object occurrence of the class. A method invocation is static if the message is statically bound to the class, i.e., the object cannot be substituted by another object and thus the class of the object and the relevant method are uniquely determined. A method invocation is dynamic if the message is dynamically bound to

[26] A child is a direct descendant, i.e., the methods and attributes of a class are directly inherited by its child classes.

the class, i.e., the class of the object is only known at run time and thus several methods are candidates for invocation. Dynamic binding is due to the polymorphism and substitutability principles. The expressions SIM(m) and PIM(m) denote respectively the set of methods that are statically and dynamically invoked by a method $m \in M(C)$. PIM(m) considers all methods that may be invoked at run time. According to the formalism of Briand *et al.*, $SIM(m) \subseteq PIM(m)$.

DEFINITION 14 (SIM(m). The Set of Statically Invoked Methods of m)

Let $c \in C$, $m \in M_I(c)$, and $m' \in M(C)$. Then $m' \in SIM(m) \Leftrightarrow \exists\, d \in C$ such that $m' \in M(d)$ and the body of m has a method invocation where m' is invoked for an object of static type class d.

DEFINITION 15 (PIM(m). The Set of Polymorphically Invoked Methods of m)

Let $c \in C$, $m \in M_I(c)$, and $m' \in M(C)$. Then $m' \in PIM(m) \Leftrightarrow \exists\, d \in C$ such that $m' \in M(d)$ and the body of m has a method invocation where m' may, because of polymorphism and dynamic binding, be invoked for an object of dynamic type class d.

A class has a set of attributes.[27] Inherited attributes are only declared in the class. Newly defined attributes are implemented in the class. Methods may reference attributes.

DEFINITION 16 (Declared and Implemented Attributes)

For each $c \in C$ let A(c) be the set of attributes of class c. $A(c) = A_D(c) \cup A_I(c)$ where

- $A_D(c)$ is the set of attributes declared in c (i.e., inherited attributes);
- $A_I(c)$ is the set of attributes implemented in c (i.e., noninherited attributes).

DEFINITION 17 (AR(m))

For each $m \in M(C)$ let AR(m) be the set of attributes referenced by method m.

Method invocations and attribute references can be used as another classification mechanism for the classes in an object-oriented system. Briand *et al.* define the 'uses' relationship by means of the predicate *Uses*. A class c 'uses' a class d if a method implemented in c invokes a method implemented in d or references an attribute implemented in d. Polymorphism is taken into account.

DEFINITION 18 (The predicate *Uses*)

Let $c \in C$, $d \in C$.

$Uses(c, d) \Leftrightarrow (\exists\, m \in M_I(c): \exists\, m' \in M_I(d): m' \in PIM(m)) \vee (\exists\, m \in M_I(c): \exists\, a \in A_I(d): a \in AR(m))$

The response set for a class consists of the methods of the class and all methods that can be invoked up to a level α of nested method invocations. Polymorphism is taken into account.

DEFINITION 19 (Response Set for a Class)

For each $c \in C$, let $\bigcup_{i=0}^{\alpha} Response_i(c)$, for $\alpha \in \{0, 1, \ldots\}$, be the response set of c.

$Response_i(c)$ is recursively defined:

$Response_0(c) = M(c)$

[27] Not to be confused with 'software attributes'.

$$Response_{i+1}(c) = \bigcup_{m \in Response_i(c)} PIM(m)$$

5.2 Applying DISTANCE

For the redefinition of the MOOSE measures we use the template shown in Table 4. In the top row we enter the original description of the measure as in [20]. The only exception is LCOM where an earlier description, found in [19], is used. The later version of LCOM, presented in [20], is a composite measure and it has repeatedly been criticised for its problematic behaviour and counter-intuitive results [35]. That is why we prefer to redefine the older, and simpler, version of LCOM.

In the second row we enter the name of the attribute of interest *attr*. This name describes the conceptual idea that reflects to some extent the empirical understanding of the attribute. However, it is only used as an identifier. The precise meaning of the attribute is contained within its distance-based definition. The software entity that is characterised by the attribute of interest is the object class, formally defined as $c \in UC$.

Next, the output of the distance-based measure construction process is described in the left column of the template. The execution of the different steps requires several choices to be made regarding the formalisation of the attributes. It was noted before that some of the MOOSE measure definitions are ambiguous and additional interpretation on the exact meaning of the attributes is required. In case of doubt we stick to the interpretations offered by Briand *et al.* in [8], [9] and by Chidamber *et al.* in [21]. We believe it is a strength of the distance-based approach that choices must be made explicit.

The right column shows the measurement theoretic constructs that underlie the different steps in the distance-based process. In fact, the right column presents the formal, distance-based, definition of the attribute and the formal, metric-based, definition of the corresponding MOOSE measure. As our interpretation of coupling, response for a class and lack of cohesion in methods is identical to the interpretation of the same attributes by Briand *et al.* [8], [9] and as the same OO formalism and terminology is used, the formal definitions of the CBO, RFC and LCOM measures are identical too. One the one hand this allows comparing our approach with the approach of Briand *et al.* On the other hand, Briand *et al.* do not offer formal definitions of the attributes. As a consequence, they do not (and cannot) show the construct validity of their formally defined MOOSE measures. In fact, Briand *et al.* evaluate the 'theoretical' validity of the measures using a property-based approach (i.e., the sets of properties presented in [13]). It is interesting to note that Chidamber *et al.* used a similar 'validation' strategy by using a set of properties proposed by Weyuker [54]. As argued before, a property-based approach is not sufficient to show the construct validity of the measures. The DISTANCE framework opts resolutely for a measurement theoretic approach. Using proximity measurement structures, the construct validity of the measures can actually be proven (cf. section 4).

A last row in the template allows entering additional remarks regarding the choices made when applying the distance-based measure construction process.

Tables 5 to 10 contain the distance-based redefinitions of the MOOSE measures. The 'remarks' section makes the tables largely self-explaining. We reiterate here again that it is not the intent of DISTANCE to provide the one and only definition of some attribute and its measure. However, the distance-based definition of the attribute and the metric-based definition of the measure allow investigating the specific notion of the attribute for which a measure with proven

construct validity has been proposed. The outcome of the different steps in the distance-based process can easily be changed to reflect other points of view.

MOOSE measure	(description of the MOOSE measure)		
Software attribute (*attr*)	(attribute of interest)	**Software entity** ($p \in P$)	(entity characterised by *attr*)
Output of distance-based process		**Underlying measurement theoretic constructs & formal definitions**	
M *abs*	(set of measurement abstractions) (abstraction function)		
T_e	(set of elementary transformation types)	$\bullet\geq$ SAPS	(empirical dissimilarity ordering) (segmentally additive proximity structure)
δ	(metric function)	MSAS	(metric space with additive segments)
ref	(reference function)	$ST_{abs(p),ref(p)}$ Attribute def.	(shortest sequences of elementary transformations) (formal distance-based attribute definition)
μ	(measure function)	Measure def.	(formal metric-based measure definition)
Remarks:			

Table 4: Template for distance-based redefinition of the MOOSE measures

<table>
<tr><td>MOOSE measure</td><td colspan="3">WMC = $\sum_{i=1}^{n} c_i$, where c_i is the static complexity value of method m_i defined in the class and n is the number of methods.
If all method complexities are considered to be unity, then WMC = n, where n is the number of methods.
(remark 1)</td></tr>
<tr><td>Software attribute
(attr)</td><td>complexity
(remark 2)</td><td>Software entity
($p \in P$)</td><td>$c \in UC$</td></tr>
<tr><td colspan="2">Output of distance-based process</td><td colspan="2">Underlying measurement theoretic constructs
&
formal definitions</td></tr>
<tr><td>M

abs</td><td>$\wp(UM)$

abs_{WMC}: $UC \rightarrow \wp(UM)$: $c \rightarrow M_I(c)$
(remark 3)</td><td></td><td></td></tr>
<tr><td>T_e</td><td>$T_{e\text{-}WMC} = \{t_{0\text{-}WMC}, t_{1\text{-}WMC}\}$,
where
$t_{0\text{-}WMC}$: $\wp(UM) \rightarrow \wp(UM)$: $s \rightarrow s \cup \{m\}$
$t_{1\text{-}WMC}$: $\wp(UM) \rightarrow \wp(UM)$: $s \rightarrow s - \{m\}$
with $m \in UM$
(remark 4)</td><td>$\bullet\geq$

SAPS</td><td>$\bullet\geq_{WMC}$

$(\wp(UM), \bullet\geq_{WMC})$</td></tr>
<tr><td>δ</td><td>δ_{WMC}: $\wp(UM) \times \wp(UM) \rightarrow \Re$
$: (s, s') \rightarrow |s - s'| + |s' - s|$
(remark 5)</td><td>MSAS</td><td>$(\wp(UM), \delta_{WMC})$</td></tr>
<tr><td>ref</td><td>ref_{WMC}: $UC \rightarrow \wp(UM)$: $c \rightarrow \emptyset$
(remark 6)</td><td>$ST_{abs(p),ref(p)}$

Attribute def.</td><td>$ST_{MI(c),\emptyset}$

The complexity of a class c is the distance between the set of methods $M_I(c)$ and the empty set.</td></tr>
<tr><td>μ</td><td>μ_{WMC}: $UC \rightarrow \Re$: $c \rightarrow |M_I(c)|$
(remark 7)</td><td>Measure def.</td><td>The complexity of a class c is measured by the count of methods in $M_I(c)$.</td></tr>
<tr><td colspan="4">Remarks:
1: We only consider the second case here. To measure the static complexity of a method, we need to know the method body, which is a construct not modelled in our OO formalism.
2: We deliberately choose for a meaningless attribute name. We may as well call this attribute 'size' or 'functionality' or whatever. The precise meaning of the attribute must be derived from its distance-based definition, and not from its name. The name only distinguishes this attribute from the other attributes captured by MOOSE.
3: Churcher et al. [22] criticise Chidamber et al. for not being more precise in what is meant by 'methods defined in the class'. In [21] Chidamber et al. respond that only methods are counted that require additional design effort. We believe this corresponds most closely to the methods implemented in a class (i.e., $M_I(c)$). Of course, other interpretations are valid as well. What matters is that the abstraction function clarifies which interpretation is adhered to.
4: s is a set of methods, i.e., an element of $\wp(UM)$. As $\wp(UM)$ is a set of sets, $T_{e\text{-}WMC}$ is constructively complete and inverse constructively complete.
5: We choose for the symmetric difference model as the form of the metric. This model defines a metric for sets.
6: This choice implies that a class with no implemented methods has zero complexity (e.g., a 'fully' abstract class, i.e., a class with only virtual methods).
7: $\forall c \in UC$: $\mu_{WMC}(c) = \delta_{WMC}(M_I(c), \emptyset) = |M_I(c) - \emptyset| + |\emptyset - M_I(c)| = |M_I(c)|$</td></tr>
</table>

Table 5: Distance-based redefinition of WMC

MOOSE measure	CBO for a class is a count of the number of other classes to which it is coupled. Two classes are coupled when methods declared in one class use methods or instance variables defined by the other class.		
Software attribute (*attr*)	coupling	**Software entity** ($p \in P$)	$c \in UC$
Output of distance-based process		**Underlying measurement theoretic constructs & formal definitions**	
M *abs*	$\wp(UC)$ $abs_{CBO}: UC \rightarrow \wp(UC)$ $: c \rightarrow \{d \in C - \{c\} \mid Uses(c, d) \vee Uses(d, c)\}$ ***(remark 1)***		
T_e	$T_{e\text{-}CBO} = \{t_{0\text{-}CBO}, t_{1\text{-}CBO}\}$ where $t_{0\text{-}CBO}: \wp(UC) \rightarrow \wp(UC): s \rightarrow s \cup \{c\}$ $t_{1\text{-}CBO}: \wp(UC) \rightarrow \wp(UC): s \rightarrow s - \{c\}$ with $c \in UC$ ***(remark 2)***	$\bullet\geq$ SAPS	$\bullet\geq_{CBO}$ $(\wp(UC), \bullet\geq_{CBO})$
δ	$\delta_{CBO}: \wp(UC) \times \wp(UC) \rightarrow \Re$ $: (s, s') \rightarrow \lvert s - s' \rvert + \lvert s' - s \rvert$ ***(remark 3)***	MSAS	$(\wp(UC), \delta_{CBO})$
ref	$ref_{CBO}: UC \rightarrow \wp(UC): c \rightarrow \varnothing$ ***(remark 4)***	$ST_{abs(p),ref(p)}$ Attribute def.	$ST_{\{d \in C - \{c\} \mid uses(c, d) \vee uses(d, c)\},\varnothing}$ The coupling of a class c is the distance between the set of classes $\{d \in C - \{c\} \mid Uses(c, d) \vee Uses(d, c)\}$ and the empty set.
μ	$\mu_{CBO}: UC \rightarrow \Re$ $: c \rightarrow \lvert \{d \in C - \{c\} \mid Uses(c, d) \vee Uses(d, c)\} \rvert$ ***(remark 5)***	Measure def.	The coupling of a class c is measured by the count of classes in $\{d \in C - \{c\} \mid Uses(c, d) \vee Uses(d, c)\}$

Remarks:

1: It is assumed that $c \in C$. The particular interpretation of coupling here is that of Briand *et al.* [9]. According to the definition of the predicate *Uses* dynamic method invocations are taken into account, but not invocations of declared methods or references to declared attributes. The abstraction function returns a set and thus each other class in C contributes at most once to the coupling of c.

2: s is a set of classes, i.e., an element of $\wp(UC)$. As $\wp(UC)$ is a set of sets, $T_{e\text{-}CBO}$ is constructively complete and inverse constructively complete.

3: The symmetric difference model defines a metric for sets.

4: Although highly unlikely, we can imagine classes that are not coupled to other classes (e.g., 'fully' abstract classes having no implemented methods, 'singleton' systems, i.e., systems containing a single class).

5: $\forall c \in UC: \mu_{CBO}(c) = \delta_{CBO}(abs_{CBO}(c), ref_{CBO}(c)) = \lvert \{d \in C - \{c\} \mid Uses(c, d) \vee Uses(d, c)\} - \varnothing \rvert + \lvert \varnothing - \{d \in C - \{c\} \mid Uses(c, d) \vee Uses(d, c)\} \rvert = \lvert \{d \in C - \{c\} \mid Uses(c, d) \vee Uses(d, c)\} \rvert$

Table 6: Distance-based redefinition of CBO

MOOSE measure	RFC = |RS| where RS is the response set for the class. The response set for the class contains all methods in the class and all methods called by these methods. ***(remark 1)***		
Software attribute (*attr*)	response for a class	**Software entity** ($p \in P$)	$c \in UC$

Output of distance-based process		**Underlying measurement theoretic constructs & formal definitions**	
M *abs*	$\wp(UM)$ $abs_{RFC}: UC \rightarrow \wp(UM)$ $: c \rightarrow \bigcup_{i=0}^{\alpha} Response_i(c)$ ***(remark 2)***		
T_e	$T_{e\text{-}RFC} = \{t_{0\text{-}RFC}, t_{1\text{-}RFC}\}$ where $t_{0\text{-}RFC}: \wp(UM) \rightarrow \wp(UM): s \rightarrow s \cup \{m\}$ $t_{1\text{-}RFC}: \wp(UM) \rightarrow \wp(UM): s \rightarrow s - \{m\}$ with $m \in UM$ ***(remark 3a)***	$\bullet\geq$ SAPS	$\bullet\geq_{RFC}$ $(\wp(UM), \bullet\geq_{RFC})$ ***(remark 3b)***
δ	$\delta_{RFC}: \wp(UM) \times \wp(UM) \rightarrow \Re$ $: (s, s') \rightarrow \|s - s'\| + \|s' - s\|$ ***(remark 4a)***	MSAS	$(\wp(UM), \delta_{RFC})$ ***(remark 4b)***
ref	$ref_{RFC}: UC \rightarrow \wp(UM): c \rightarrow \varnothing$ ***(remark 5)***	$ST_{abs(p),ref(p)}$ Attribute def.	$ST \bigcup_{i=0}^{\alpha} Response_{i(c),\varnothing}$ The response for a class c is the distance between the set of methods $\bigcup_{i=0}^{\alpha} Response_i(c)$ and the empty set.
μ	$\mu_{RFC}: UC \rightarrow \Re: c \rightarrow \|\bigcup_{i=0}^{\alpha} Response_i(c)\|$ ***(remark 6)***	Measure def.	The response for a class c is measured by the count of methods in $\bigcup_{i=0}^{\alpha} Response_i(c)$

Remarks:

1: In [20] membership to the response set is defined only up to the first level of nesting of method calls. We will work with the RFC_α variant of the measure, whose definition can be instantiated to any level α of nested method invocations. This variant has been suggested by Churcher *et al.* [23] and formalised by Briand *et al.* [9].

2: The level α must be set. The abstraction function returns the response set of the class up to the level α. Our definition of the response set was taken from [9] to ensure an identical interpretation (e.g., dynamic method invocations are taken into account).

3a&b: $T_{e\text{-}RFC}$ is identical to $T_{e\text{-}WMC}$ as a constructively complete and inverse constructively complete set of elementary transformations for $\wp(UM)$ must be found. Consequently, $(\wp(UM), \bullet\geq_{RFC})$ and $(\wp(UM), \bullet\geq_{WMC})$ denote the same proximity structure.

4a&b: δ_{RFC} is identical to δ_{WMC} and thus $(\wp(UM), \delta_{RFC})$ and $(\wp(UM), \delta_{WMC})$ denote the same metric space.

5: This choice assumes the existence of a hypothetical 'empty' class having no methods.

6: $\forall c \in UC: \mu_{RFC}(c) = \delta_{RFC}(abs_{RFC}(c), ref_{RFC}(c)) = |\bigcup_{i=0}^{\alpha} Response_i(c) - \varnothing| + |\varnothing - \bigcup_{i=0}^{\alpha} Response_i(c)| = |\bigcup_{i=0}^{\alpha} Response_i(c)|$

Table 7: Distance-based redefinition of RFC

MOOSE measure	LCOM is the number of disjoint sets formed by the intersection of the n sets $\{I_i\}$, where n is the number of methods in the class and $\{I_i\}$ is the set of instance variables used by method m_i. ***(remark 1)***		
Software attribute (*attr*)	lack of cohesion in methods	**Software entity** ($p \in P$)	$c \in UC$

Output of distance-based process		**Underlying measurement theoretic constructs & formal definitions**	
M *abs*	$\wp(UM \times UM)$ $abs_{LCOM}: UC \rightarrow \wp(UM \times UM)$ $: c \rightarrow \{(m, m') \in M_I(c) \times M_I(c) \mid m \neq m' \wedge AR(m) \cap AR(m') \cap A_I(c) = \varnothing\}$ ***(remark 2)***		
T_e	$T_{e\text{-}LCOM} = \{t_{0\text{-}LCOM}, t_{1\text{-}LCOM}\}$ where $t_{0\text{-}LCOM}: \wp(UM \times UM) \rightarrow \wp(UM \times UM)$ $: s \rightarrow s \cup \{(m, m')\}$ $t_{1\text{-}LCOM}: \wp(UM \times UM) \rightarrow \wp(UM \times UM)$ $: s \rightarrow s - \{(m, m')\}$ with $(m, m') \in UM \times UM$ ***(remark 3)***	$\bullet\geq$ SAPS	$\bullet\geq_{LCOM}$ $(\wp(UM \times UM), \bullet\geq_{LCOM})$
δ	$\delta_{LCOM}: \wp(UM \times UM) \times \wp(UM \times UM) \rightarrow \Re$ $: (s, s') \rightarrow \lvert s - s' \rvert + \lvert s' - s \rvert$ ***(remark 4)***	MSAS	$(\wp(UM \times UM), \delta_{LCOM})$
ref	$ref_{LCOM}: UC \rightarrow \wp(UM \times UM): c \rightarrow \varnothing$ ***(remark 5)***	$ST_{abs(p),ref(p)}$ Attribute def.	$ST_{\{(m, m') \in MI(c) \times MI(c) \mid m \neq m' \wedge AR(m) \cap AR(m') \cap AI(c) = \varnothing\},\varnothing}$ The lack of cohesion in methods of a class c is the distance between the set of pairs of methods $\{(m, m') \in M_I(c) \times M_I(c) \mid m \neq m' \wedge AR(m) \cap AR(m') \cap A_I(c) = \varnothing\}$ and the empty set.
μ	$\mu_{LCOM}: UC \rightarrow \Re$ $: c \rightarrow \lvert \{(m, m') \in M_I(c) \times M_I(c) \mid m \neq m' \wedge AR(m) \cap AR(m') \cap A_I(c) = \varnothing\} \rvert$ ***(remark 6)***	Measure def.	The lack of cohesion in methods of a class c is measured by the count of pairs of methods in $\{(m, m') \in M_I(c) \times M_I(c) \mid m \neq m' \wedge AR(m) \cap AR(m') \cap A_I(c) = \varnothing\}$

Remarks:

1: This is the LCOM version described in [19]. According to Henderson-Sellers [34] this description is meaningless as the intersection of n sets is always a single set. Briand *et al.* [8] interpret this LCOM version as the count of method pairs having no common attribute references.

2: The abstraction function returns for a class c a set of method pairs. The methods in such a pair must be different, must be implemented in the class, and may not have common references to attributes implemented in c [8]. Assume further that the pairs (m, m') and (m', m) are identical and thus considered a single element (i.e., the ordering of the methods in a pair is of no importance).

3: s is a set of pairs of methods, i.e., an element of $\wp(UM \times UM)$. As $\wp(UM \times UM)$ is a set of sets, $T_{e\text{-}LCOM}$ is constructively complete and inverse constructively complete.

4: The symmetric difference model defines a metric for sets.

5: If all possible pairs of implemented methods have (implemented) attribute references in common, then there is no lack of cohesion in methods.

6: $\forall c \in UC: \mu_{LCOM}(c) = \delta_{LCOM}(abs_{LCOM}(c), ref_{LCOM}(c)) = \lvert \{(m, m') \in M_I(c) \times M_I(c) \mid m \neq m' \wedge AR(m) \cap AR(m') \cap A_I(c) = \varnothing\} - \varnothing \rvert + \lvert \varnothing - \{(m, m') \in M_I(c) \times M_I(c) \mid m \neq m' \wedge AR(m) \cap AR(m') \cap A_I(c) = \varnothing\} \rvert = \lvert \{(m, m') \in M_I(c) \times M_I(c) \mid m \neq m' \wedge AR(m) \cap AR(m') \cap A_I(c) = \varnothing\} \rvert$

MOOSE measure	NOC is the number of immediate subclasses subordinated to a class in the class hierarchy		
Software attribute (*attr*)	numerousness of children	**Software entity** ($p \in P$)	$c \in UC$
Output of distance-based process		**Underlying measurement theoretic constructs & formal definitions**	
M *abs*	$\wp(UC)$ $abs_{NOC}: UC \rightarrow \wp(UC): c \rightarrow Children(c)$ ***(remark 1)***		
T_e	$T_{e\text{-}NOC} = \{t_{0\text{-}NOC}, t_{1\text{-}NOC}\}$ where $t_{0\text{-}NOC}: \wp(UC) \rightarrow \wp(UC): s \rightarrow s \cup \{c\}$ $t_{1\text{-}NOC}: \wp(UC) \rightarrow \wp(UC): s \rightarrow s - \{c\}$ with $c \in UC$ ***(remark 2a)***	$\bullet\geq$ SAPS	$\bullet\geq_{NOC}$ $(\wp(UC), \bullet\geq_{NOC})$ ***(remark 2b)***
δ	$\delta_{NOC}: \wp(UC) \times \wp(UC) \rightarrow \Re$ $: (s, s') \rightarrow \lvert s - s' \rvert + \lvert s' - s \rvert$ ***(remark 3a)***	MSAS	$(\wp(UC), \delta_{NOC})$ ***(remark 3b)***
ref	$ref_{NOC}: UC \rightarrow \wp(UC): c \rightarrow \varnothing$ ***(remark 4)***	$ST_{abs(p),ref(p)}$ Attribute def.	$ST_{Children(c),\varnothing}$ The numerousness of children of a class c is the distance between the set of classes *Children*(c) and the empty set.
μ	$\mu_{NOC}: UC \rightarrow \Re: c \rightarrow \lvert Children(c) \rvert$ ***(remark 5)***	Measure def.	The numerousness of children of a class c is measured by the count of classes in *Children*(c).

Remarks:

1: It is assumed that $c \in C$.

2a&b: $T_{e\text{-}NOC}$ is identical to $T_{e\text{-}CBO}$. $(\wp(UC), \bullet\geq_{NOC})$ and $(\wp(UC), \bullet\geq_{CBO})$ denote the same proximity structure.

3a&b: δ_{NOC} is identical to δ_{CBO}. $(\wp(UC), \delta_{NOC})$ and $(\wp(UC), \delta_{CBO})$ denote the same metric space.

4: We can easily imagine classes having no children. In fact, such classes exist in every object-oriented system.

5: $\forall c \in UC: \mu_{NOC}(c) = \delta_{NOC}(abs_{NOC}(c), ref_{NOC}(c)) = \lvert Children(c) - \varnothing \rvert + \lvert \varnothing - Children(c) \rvert = \lvert Children(c) \rvert$

Table 9: Distance-based redefinition of NOC

MOOSE measure	DIT is the maximum length from the node (representing the class in the inheritance tree) to the root of the tree.		
Software attribute (*attr*)	depth in inheritance graph ***(remark 1)***	**Software entity** ($p \in P$)	$c \in UC$
Output of distance-based process		**Underlying measurement theoretic constructs & formal definitions**	
M *abs*	*Nat* abs_{DIT}: UC → *Nat*: : c → max[i over all $Level_i$(C) that contain c] ***(remark 2)***		
T_e	$T_{e\text{-}DIT} = \{t_{0\text{-}DIT}, t_{1\text{-}DIT}\}$ where $t_{0\text{-}DIT}$: *Nat* → *Nat*: i → i + 1 $t_{1\text{-}DIT}$: *Nat* → *Nat*: i → i - 1 ***(remark 3)***	$\bullet\geq$ SAPS	$\bullet\geq_{DIT}$ (*Nat*, $\bullet\geq_{DIT}$)
δ	δ_{DIT}: *Nat* × *Nat* → $\Re$: (i,j) → $\lvert i - j \rvert$ ***(remark 4)***	MSAS	(*Nat*, δ_{DIT})
ref	ref_{DIT}: UC → *Nat*: c → 0 ***(remark 5)***	$ST_{abs(p),ref(p)}$ Attribute def.	$ST_{max[i\ over\ all\ Level_i(C)\ that\ contain\ c],0}$ The depth in inheritance graph of a class c is the distance between the natural number max[i over all $Level_i$(C) that contain c] and 0.
μ	μ_{DIT}: UC → $\Re$: c → max[i over all $Level_i$(C) that contain c] ***(remark 6)***	Measure def.	The depth in inheritance graph of a class c is measured by the natural number max[i over all $Level_i$(C) that contain c].

Remarks:

1: We prefer to refer to an inheritance graph (instead of a tree) as the inheritance relation does not necessarily define a hierarchy on the classes in a system.

2: Assume that $c \in C$. The measurement abstraction is chosen to be a natural number representing the deepest level in the inheritance graph that contains the class. This interpretation corresponds to the informal definition of DIT by Chidamber *et al.*

3: It is easily shown that these elementary transformation types are sufficient to turn the set of natural numbers into a proximity structure - a proof is omitted here for brevity's sake. Note that the empirical relational structure is here a numerical relational structure as well. However, the numerical relational structure used to map measurement values into is based on the set of real numbers (because of the admissible transformations of scale, i.e., the multiplication by a positive real constant c).

4: The absolute difference between two natural numbers is a metric value for the distance between these numbers - a proof is omitted here for brevity's sake.

5: Root classes have no depth in the inheritance tree. They are found at level 0.

6: $\forall c \in UC$: $\mu_{DIT}(c) = \delta_{DIT}(abs_{DIT}(c), ref_{DIT}(c)) = \lvert$max[i over all $Level_i$(C) that contain c]- 0$\rvert$ = max[i over all $Level_i$(C) that contain c]

Table 10: Distance-based redefinition of DIT

6 CONCLUSIONS

In this paper a new framework for software measurement is presented. DISTANCE uses the easily imagined, detected and visualised concepts of similarity and dissimilarity between software entities with respect to some relevant characteristic(s), as a basis for a formal definition of both the software attribute of interest and the software measure. We call the attribute distance-based defined. An attribute characterising a software entity is defined as the distance (or dissimilarity) between, on the one hand a model of the entity that emphasises the attribute, and on the other hand a reference model for the attribute. The measure of the attribute is called metric-based defined. The metric value for the distance between model and reference model is used as a measurement value for the attribute of interest.

The basic features of the distance-based measurement approach as compared to related work are the following:

- The approach is sufficient to prove the construct validity of the resulting measures. The approach is consistent with proximity measurement, which is a measurement structure specifically suited for the measurement of distance or dissimilarity. DISTANCE thus builds upon well-founded measurement theoretic constructs and theorems.
- The resulting measures are characterised by the ratio scale type.[28]
- Constructive procedures are offered both to define the attribute of interest and to define the actual software measure. These step-by-step procedures hide the complexity of the underlying measurement theoretic representation from the users of the approach.
- The approach is systematic in the sense that DISTANCE contains a process model for measure construction. The process model details for each task inputs, outputs and underlying assumptions. It prescribes the order of execution using an UML activity diagram. Finally, it acts as a 'measure construction component' that can be embedded into a typical goal-oriented measurement approach for empirical software engineering research (e.g., GQM).

Some of the related work, both measurement theoretic and property-based approaches to software measurement, is also characterised by some of the above features, but not by all of them simultaneously. The contribution of DISTANCE to the field is that it is at the same time a formal, constructive and systematic approach. Moreover, the application of DISTANCE to object-oriented software measures showed that results can be obtained that were not possible with the current approaches (e.g., the belief function approach of Zuse [58] does not support the ratio scale type).

Although the distance-based approach compares favourably to related work, it is based on a number of underlying assumptions that were explicitly identified in the paper. Whereas we did not claim that these assumptions hold in each and every case, we believe that the approach is generic and flexible enough to be of practical use. Our initial experiences with the approach relate mainly to the measurement of object-oriented software [47]. Future research will focus on the application of DISTANCE in the context of component-oriented software engineering.

---

[28] In [47] we showed that DISTANCE also supports the ordinal scale type. In fact, only the underlying measurement theoretic representation must be changed. It is sufficient to use a representation by an 'ordinary' metric, but not a metric with additive segments. Also, the proximity structure is no longer required to be segmentally additive. Showing that the constructive procedures result in a representation by a metric with additive segments also shows that they result in a representation by an 'ordinary' metric. Hence, the procedures of DISTANCE need not to be changed. A further discussion is beyond the scope of this paper.

REFERENCES

[1] A.L. Baker, J.M. Bieman, N. Fenton, D.A. Gustafson, A. Melton, and R. Whitty, "A Philosophy for Software Measurement", *J. Systems and Software*, vol. 12, no. 3, pp. 277-281, Mar. 1990.

[2] V.R. Basili, L. Briand, and W.L. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators", *IEEE Trans. Software Eng.*, vol. 22, no. 10, pp. 751-761, Oct. 1996.

[3] V.R. Basili, G. Caldiera, and H.D. Rombach, "The Goal Question Metric Approach", in: Marciniak (ed.), *Encyclopedia of Software Engineering*, Wiley, 1994.

[4] V.R. Basili and H.R. Rombach, "The TAME project: towards improvement-oriented software environments", *IEEE Trans. Software Eng.*, vol. 14, no. 6, pp. 728-738, June 1988.

[5] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.

[6] L. Briand, E. Arisholm, S. Counsell, F. Houdek, and P. Thévenod-Fosse, "Empirical Studies of Object-Oriented Artifacts, Methods, and Processes: State of The Art and Future Directions", Tech. Report ISERN-99-12 / IESE 037.99/E, Fraunhofer Inst. for Experimental Software Eng., Germany, 1999, 12 pp.

[7] L.C. Briand, J.W. Daly, V. Porter, and J. Wüst, "A Comprehensive Empirical Validation of Product Measures for Object-Oriented Systems", Tech. Report ISERN-98-07, Fraunhofer Inst. For Experimental Software Eng., Germany, 1998, 35 pp.

[8] L.C. Briand, J.W. Daly, and J. Wüst, "A Unified Framework for Cohesion Measurement in Object-Oriented Systems", *Empirical Software Eng., An Int'l J.*, vol. 3, no. 1, pp. 65-117, 1998.

[9] L.C. Briand, J.W. Daly, and J.K. Wüst, "A Unified Framework for Coupling Measurement in Object-Oriented Systems", *IEEE Trans. Software Eng.*, vol. 25, no. 1, pp. 91-121, Jan.-Feb. 1999.

[10] L. Briand, K. El Emam, and S. Morasca, "Theoretical and Empirical Validation of Software Product Measures", Tech. Report ISERN-95-03, International Software Engineering Network, 1995, 23 pp.

[11] L. Briand, K. El Emam, and S. Morasca, "On the Application of Measurement Theory in Software Engineering", *Empirical Software Eng., An Int'l J.*, vol. 1, no. 1, pp. 61-88, 1996.

[12] L. Briand, S. Morasca, and V.R. Basili, "Goal-Driven Definition of Product Metrics Based on Properties", Tech. Report CS-TR-3346, Computer Science Dept., University of Maryland, MD, September 1994, 20 pp.

[13] L.C. Briand, S. Morasca, and V.R. Basili, "Property-Based Software Engineering Measurement", *IEEE Trans. Software Eng.*, vol. 22, no. 1, pp. 68-86, Jan. 1996.

[14] L.C. Briand, S. Morasca, and V.R. Basili, "An Operational Process for Goal-Driven Definition of Measures", Tech. Report ISERN-99-04 / IESE 017.99/E, Fraunhofer Inst. For Experimental Software Eng., Germany, 1999, 30 pp.

[15] L.C. Briand, J. Wüst, S. Ikonomovski, and H. Lounis, "A Comprehensive Investigation of Quality Factors in Object-Oriented Designs: an Industrial Case Study", *Proc. 21st Int'l Conf. Software Eng.*, Los Angeles, CA, pp. 345-354, 1999.

[16] S.S. Brilliant and J.C. Knight, "Final Report of the Empirical Research in Software Engineering Workshop", *ACM SIGSOFT Software Eng. Notes*, vol. 24, no. 3, pp. 45-52, May 1999.

[17] M. Cartwright and M. Shepperd, "An Empirical View of Inheritance", Tech. Report ESERG: TR98-002, Dept. Computing, Bournemouth University, UK, 1998, 11 pp.

[18] S.R. Chidamber, D.P. Darcy, and C.F. Kemerer, "Managerial Use of Metrics for Object-Oriented Software", *IEEE Trans. Software Eng.*, vol. 24, no. 8, pp. 629-639, Aug. 1998.

[19] S.R. Chidamber and C.F. Kemerer, "Towards a Metrics Suite for Object Oriented Design", *Proc. 6th ACM Conf. Object Oriented Programming, Languages, and Applications*, Phoenix, AZ, pp. 197-211, 1991.

[20] S.R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object Oriented Design", *IEEE Trans. Software Eng.*, vol. 20, no. 6, pp. 476-493, June 1994.

[21] S.R. Chidamber and C.F. Kemerer, "Authors' Reply to Comments on A Metrics Suite for Object Oriented Design", *IEEE Trans. Software Eng.*, vol. 21, no. 3, p. 265, Mar. 1995.

[22] N.I. Churcher and M.J. Shepperd, "Comments on A Metrics Suite for Object Oriented Design", *IEEE Trans. Software Eng.*, vol. 21, no. 3, pp. 263-265, Mar. 1995.

[23] N.I. Churcher and M.J. Shepperd, "Towards a Conceptual Framework for Object Oriented Software Metrics", *ACM SIGSOFT Software Eng. Notes*, vol. 20, no. 2, pp. 69-76, Apr. 1995.

[24] J. Daly, A. Brooks, J. Miller, M. Roper, and M. Wood, "An Empirical Study Evaluating Depth of Inheritance on the Maintainability of Object-Oriented Software", *Empirical Software Eng.: An Int'l J.*, vol. 1, no. 2, pp. 109-132, 1996.

[25] P. Devanbu, S. Karstu, W. Melo, and W. Thomas, "Analytical and Empirical Evaluation of Software Reuse Metrics", *Proc. 18th Int'l Conf. Software Eng.*, Berlin, 1996.

[26] N.E. Fenton, "Software metrics: theory, tools and validation", *Software Eng. J.*, vol. 5, no. 1, pp. 65-78, Jan. 1990.

[27] N.E. Fenton, *Software Metrics: A Rigorous Approach*, Chapman & Hall, London, 1991, 337 pp.

[28] N. Fenton, "When a Software Measure is Not a Measure", *IEE Software Eng. J.*, vol. 7, no. 5, pp. 357-362, Sept. 1992.

[29] N. Fenton, "Software Measurement: A Necessary Scientific Basis", *IEEE Trans. Software Eng.*, vol. 20, no. 3, pp. 199-206, Mar. 1994.

[30] N.E. Fenton and S.L. Pfleeger, *Software Metrics: A Rigorous & Practical Approach*, 2nd edition, PWS Publishing Company, London, 1997, 638 pp.

[31] N. Fenton, S.L. Pfleeger, and R.L. Glass, Science and Substance: A Challenge to Software Engineers", *IEEE Software*, vol. 11, no. 4, pp. 86-95, July 1994.

[32] D.A. Gustafson, J.T. Tan, and P. Weaver, "Software Measure Specification", *ACM SIGSOFT Software Eng. Notes*, vol. 18, no. 5, pp. 163-168, Dec. 1993.

[33] R. Harrison, S. Counsell, and R. Nithi, "Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems", *Proc. 3rd Int'l Conf. Empirical Assessment & Evaluation in Software Eng.*, Keele, UK, Apr. 1999.

[34] B. Henderson-Sellers, "The Mathematical Validity of Software Metrics", *ACM SIGSOFT Software Eng. Notes*, vol. 21, no. 5, pp. 89-94, Sept. 1996.

[35] M. Hitz and B. Montazeri, "Chidamber and Kemerer's Metrics Suite: A Measurement Theory Perspective", IEEE Trans. Software Eng., vol. 22, no. 4, pp. 267-271, Apr. 1996.

[36] B. Kitchenham, S.L. Pfleeger, and N. Fenton, "Towards a Framework for Software Measurement Validation", *IEEE Trans. Software Eng.*, vol. 21, no. 12, pp. 929-944, Dec. 1995.

[37] B.A. Kitchenham and J.G. Stell, "The Danger of Using Axioms in Software Metrics", *IEE Proc. Software Eng.*, vol. 144, no. 5-6, pp. 279-285, Oct.-Dec. 1997.

[38] D.H. Krantz, R.D. Luce, P. Suppes, and A. Tversky, *Foundations of Measurement*, vol. 1, Academic Press, New York, 1971, 557 pp.

[39] K.B. Lakshmanan, S. Jayaprakash, and P.K. Sinha, "Properties of Control-Flow Complexity Measures", *IEEE Trans. Software Eng.*, vol. 17, no. 12, pp. 1289-1295, Dec. 1991.

[40] W. Li and S. Henry, "Object-Oriented Metrics that Predict Maintainability", *J. Systems and Software*, vol. 23, no. 2, pp. 111-122, Nov. 1993.

[41] Y. Mao, H.A. Sahraoui, and H. Lounis, "Impact of Complexity on Reusability in OO Systems", *Proc. OO Product Metrics for Software Quality Assessment Workshop*, Brussels, pp. 60-65, July 1998.

[42] A.C. Melton, D.A. Gustafson, J.M. Bieman, and A.L. Baker, "A Mathematical Perspective for Software Measures Research", *IEE Software Eng. J.*, vol. 5, no. 5, pp. 246-254, Sept. 1990.

[43] M.G. Mendonça, V.R. Basili, I.S. Bhandari, and J. Dawson, "An approach to improving existing measurement frameworks", *IBM Systems J.*, vol. 37, no. 4, pp. 484-501, 1998.

[44] S. Morasca and L.C. Briand, "Towards a Theoretical Framework for Measuring Software Attributes", *Proc. 4th Int'l Symp. Software Metrics*, Albuquerque, NM, Nov. 1997.

[45] S. Morasca, L.C. Briand, V.R. Basili, E.J. Weyuker, and M.V. Zelkowitz, "Comments on Towards a Framework for Software Measurement Validation", *IEEE Trans. Software Eng.*, vol. 23, no. 3, pp. 187-188, Mar. 1997.

[46] B.J. Oommen, K. Zhang, and W. Lee, "Numerical Similarity and Dissimilarity Measures Between Two Trees", *IEEE Trans. Computers*, vol. 45, no. 12, pp. 1426-1434, Dec. 1996.

[47] G. Poels, *On the Formal Aspects of the Measurement of Object-Oriented Software Specifications*, Ph.D. Dissertation, Catholic University of Leuven, Belgium, Apr. 1999, 507 pp.

[48] G. Poels and G. Dedene, "Distance-based software measurement: necessary and sufficient properties for software measures", 1999, accepted for publication in *Information and Software Technology*.

[49] F.S. Roberts, *Measurement Theory with Applications to Decisionmaking, Utility, and the Social Sciences*, Addison-Wesley, Reading, Mass., 1979, 420 pp.

[50] M. Shepperd and M. Cartwright, "An Empirical Investigation of an Object-Oriented System", Tech. Report, Dept. Computing, Bournemouth University, UK, Oct. 1997, 20 pp.
[51] M. Snoeck and G. Dedene, "Existence Dependency: The Key to Semantic Integrity Between Structural and Behavioural Aspects of Object Types', *IEEE Trans. Software Eng.*, vol. 24, no. 4, pp. 233-251, Apr. 1998.
[52] P. Suppes, D.M. Krantz, R.D. Luce, and A. Tversky, *Foundations of Measurement: Geometrical, Threshold, and Probabilistic Representations*, vol. 2, Academic Press, San Diego, Calif., 1989, 493 pp.
[53] J. Tian and M.V. Zelkowitz, "A Formal Program Complexity Model and Its Application", *J. Systems and Software*, vol. 17, no. 3, pp. 253-266, Mar. 1992.
[54] E.J. Weyuker, "Evaluating Software Complexity Measures", *IEEE Trans. Software Eng.*, vol. 14, no. 9, pp. 1357-1365, Sept. 1988.
[55] S.A. Whitmire, *Object Oriented Design Measurement*, Wiley Computer Publishing, New York, 1997, 452 pp.
[56] M.V. Zelkowitz and D. Wallace, "Experimental Validation in Software Engineering", *Proc. 1st Int'l Conf. Empirical Assessment & Evaluation in Software Eng.*, Keele, UK, Mar. 1997, 14 pp.
[57] K. Zhang and D. Shasha, "Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems', *Siam J. Computing*, vol. 18, no. 6, pp. 1245-1262, 1989.
[58] H. Zuse, "Foundations of Object-Oriented Software Measures", *Proc. 3rd Int'l Symp. Software Metrics*, Berlin, Mar. 1996.
[59] H. Zuse, *A Framework for Software Measurement*, Walter de Gruyter, Berlin, 1998, 755 pp.
[60] H. Zuse and P. Bollmann, "Software Metrics: Using Measurement Theory to Describe the Properties and Scales of Static Software Complexity Metrics", *ACM SIGPLAN Notices*, vol. 24, no. 8, pp. 23-33, Aug. 1989.

APPENDIX A: VERIFICATION OF THE EMPIRICAL CONDITIONS

Let M be a set of software entities that can be used as measurement abstractions of the software entities in a set P for some attribute of interest. Let $T_e = \{t_0, t_1, \ldots, t_n\}$ be a constructively complete and inverse constructively complete set of elementary transformation types for M. Define an order $\bullet\geq \subseteq M^4$ such that $\forall\ m_1, m_2, m_3, m_4 \in M$: $(m_1, m_2) \bullet\geq (m_3, m_4)$ denotes the observation that the length of the sequences in $ST_{m1,m2}$ is greater or equal than the length of the sequences in $ST_{m3,m4}$. Let $\approx$ and $\bullet>$ denote respectively the equivalence relation and the strict ordering relation derived from $\bullet\geq$ (Def. 6).

In this appendix we show that $(M, \bullet\geq)$ is a segmentally additive proximity structure.

Empirical condition S_1 of the segmentally additive proximity structure (Def. 7) refers to the conditions P_1 to P_4 of the proximity structure (Def. 6). The observation criterion for the dissimilarity ordering $\bullet\geq$ on M allows verifying these properties.

Property P_1. $\bullet\geq$ is a weak order, i.e.,

- transitive ($\forall\ m_1, m_2, m_3, m_4, m_5, m_6 \in M$: $(m_1, m_2) \bullet\geq (m_3, m_4)$ and $(m_3, m_4) \bullet\geq (m_5, m_6) \Rightarrow (m_1, m_2) \bullet\geq (m_5, m_6)$)
- strongly complete ($\forall\ m_1, m_2, m_3, m_4 \in M$: $(m_1, m_2) \bullet\geq (m_3, m_4)$ or $(m_3, m_4) \bullet\geq (m_1, m_2)$)

Verification

Transitivity

Let $m_1, m_2, m_3, m_4, m_5, m_6 \in M$. Suppose the length of the sequences in $ST_{m1,m2}$, $ST_{m3,m4}$ and $ST_{m5,m6}$ is respectively k, k' and k". Then, $(m_1, m_2) \bullet\geq (m_3, m_4)$ implies $k \geq k'$ and $(m_3, m_4) \bullet\geq (m_5, m_6)$ implies $k' \geq k"$. As the $\geq$ relation is transitive for natural numbers, it then holds that $k \geq k"$, leading to the observation that $(m_1, m_2) \bullet\geq (m_5, m_6)$.

Strong completeness

As T_e is constructively complete and inverse constructively complete, there is at least one shortest sequence of elementary transformations between any two elements of M. As all sequences of elementary transformations have a length represented by the natural

number k, and any pair of natural numbers can be compared using ≥, strong completeness is shown.

Property P_2. $\forall$ m, m' ∈ M: (m, m') •> (m, m) whenever m ≠ m' (i.e., positivity)

Verification

The length of the sequences in $ST_{m,m}$ is always zero. No transformations are required. On the other hand, when m ≠ m', then at least one elementary transformation is required to transform m into m'. Hence, the length of the sequences in $ST_{m,m'}$ is always positive. As a consequence, (m, m') •> (m, m)

Property P_3. $\forall$ m, m' ∈ M: (m, m) ≈ (m', m') (i.e., minimality)

Verification

If no transformations are required to transform m into m and m' into m', then no difference in the length of the shortest sequences can be observed. Hence, (m, m) ≈ (m', m').

To verify the symmetry property (i.e., P_4) we need to introduce the function *Inverse* that 'inverses' a sequence of elementary transformations. If a sequence of elementary transformations $t_{i1}, \ldots, t_{ik}$ takes m ∈ M to m' ∈ M, then *Inverse*($t_{i1}, \ldots, t_{ik}$) can be used to take m' back to m.

DEFINITION 20 (The function *Inverse*)

Inverse: $\boldsymbol{T} \rightarrow \boldsymbol{T}$: $t_{i1}, t_{i2}, \ldots, t_{ik-1}, t_{ik} \rightarrow t_{ik}, t_{ik-1}, \ldots, t_{i2}, t_{i1}$

Property P_4. $\forall$ m, m' ∈ M: (m, m') ≈ (m', m) (i.e., symmetry)

Verification

For any $T_{m,m'} \in ST_{m,m'}$, *Inverse*($T_{m,m'}$) ∈ $ST_{m',m}$ and the length of the sequences in $ST_{m,m'}$ and $ST_{m',m}$ is equal. If *Inverse*($T_{m,m'}$) would be longer than some $T_{m',m} \in ST_{m',m}$, then *Inverse*($T_{m',m}$) would be shorter than $T_{m,m'}$, which would imply that $T_{m,m'} \notin ST_{m,m'}$.

Empirical condition S_2 of the segmentally additive proximity structure (Def. 7) is the segmental solvability condition. The specific construction of the proximity structure by means of a constructively complete and inverse constructively complete set of elementary transformation types allows verifying this property.

Property S_2. $\forall$ m_1, m_2, m_3, m_4 ∈ M: (m_1, m_2) •≥ (m_3, m_4)
⇒ ∃ m_5 ∈ M: $(m_1, m_5) \approx (m_3, m_4)$ and $\langle m_1 m_5 m_2 \rangle$.

Verification

Suppose (m_1, m_2) •≥ (m_3, m_4) and let the length of the sequences in $ST_{m3,m4}$ be k. Then, we can always find an abstraction m_5 ∈ M such that $\langle m_1 m_5 m_2 \rangle$ and the length of the sequences in $ST_{m1,m5}$ is k. Take any $T_{m1,m2} \in ST_{m1,m2}$ and m_5 is the abstraction m_k of the T-derivation defined by $T_{m1,m2}$.

According to Whitmire [55], for many types of empirical relational structure the Archimedean axiom (S_3) is the most difficult empirical condition to test. Informally, S_3 implies that when m_3 and m_4 are dissimilar, this dissimilarity cannot be infinitely small as compared to the dissimilarity of other pairs of elements.

Property S_3. $\forall$ m_1, m_2, m_3, m_4 ∈ M: $m_3 \neq m_4$
⇒ ∃ $e_0', \ldots, e_n'$ ∈ M such that $e_0' = m_1$, $e_n' = m_2$
and (m_3, m_4) •≥ (e_{i-1}', e_i'), for all i = 1, …, n.

Verification

The smallest positive dissimilarity that m_3 and m_4 can have is when they can be transformed into each other by means of a single elementary transformation. The length of such a shortest sequence is one, which is not infinitely small as compared to any other natural number that represents the length of a shortest sequence of elementary transformations.

APPENDIX B: VERIFICATION OF THE NUMERICAL CONDITIONS

Let M be a set of software entities that can be used as measurement abstractions of the software entities in a set P for some attribute of interest. Let $T_e = \{t_0, t_1, \ldots, t_n\}$ be a constructively and inverse constructively complete set of elementary transformation types for M. Define φ: $\boldsymbol{T} \rightarrow \Re$: $T_{m,m'} \rightarrow$ k.c, where k is the length of $T_{m,m'}$ and c is a positive real number. Define δ: $M \times M \rightarrow \Re$: $(m, m') \rightarrow \varphi(T_{m,m'})$, where $T_{m,m'} \in \boldsymbol{ST_{m,m'}}$.

It can be shown that δ satisfies the representation condition of the representation theorem for segmentally additive proximity structures (i.e., properties M_1 to M_3 in Theorem 1).

Property M_1 requires (M, δ) to be a metric space. Hence, the metric axioms A_1 to A_4 (Def. 1) must be verified.

Property A_1. $\forall$ m, m' $\in$ M: $\delta(m, m') \geq 0$ (i.e., non-negativity)
Verification

$\forall$ m, m' $\in$ M: c > 0 and k $\geq$ 0 $\Rightarrow$ $\delta(m, m') \geq 0$

Property A_2. $\forall$ m, m' $\in$ M: $\delta(m, m') = 0 \Leftrightarrow m = m'$ (i.e., identity)
Verification

$\Rightarrow$ *implication*

$\delta(m, m') = 0 \Rightarrow \varphi(T_{m,m'}) = 0$ and $c > 0 \Rightarrow k = 0 \Rightarrow m = m'$

$\Leftarrow$ *implication*

$m = m' \Rightarrow \boldsymbol{ST_{m,m'}} = \{\varnothing\} \Rightarrow \delta(m, m') = \varphi(\varnothing) = 0.c = 0$

To verify the symmetry property (i.e., A_3) we use the *Inverse* function (Def. 20).

Property A_3. $\forall$ m, m' $\in$ M: $\delta(m, m') = \delta(m', m)$ (i.e., symmetry)
Verification

Let $T_{m,m'} \in \boldsymbol{ST_{m,m'}}$ and let the length of $T_{m,m'}$ be k. As $\mathit{Inverse}(T_{m,m'}) \in \boldsymbol{ST_{m',m}}$, it holds that $\delta(m, m') = \varphi(T_{m,m'}) = k.c = \varphi(\mathit{Inverse}(T_{m,m'})) = \delta(m', m)$

To verify the triangle inequality (i.e., A_4) we first need to define the partial function $\oplus$: $\boldsymbol{T} \times \boldsymbol{T} \rightarrow \boldsymbol{T}$ that is used to combine sequences of elementary transformations.

DEFINITION 21 (The partial function $\oplus$)

Let $T_{m,m'} \in \boldsymbol{T_{m,m'}}$ be a sequence of elementary transformations $t_{i1}, \ldots, t_{ik}$ that defines the T-derivation $m_0, m_1, \ldots, m_{k-1}, m_k$ from $m = m_0$ to $m' = m_k$, where $t_{ij}(m_{j-1}) = m_j$ for $1 \leq j \leq k$. Let $T_{m',m''} \in \boldsymbol{T_{m',m''}}$ be a sequence of elementary transformations $t_{ik+1}, \ldots, t_{ih}$ that defines the T-derivation $m_k, m_{k+1}, \ldots, m_{h-1}, m_h$ from $m' = m_k$ to $m'' = m_h$, where $t_{ij}(m_{j-1}) = m_j$ for $k < j \leq h$. Then, $\oplus(T_{m,m'}, T_{m',m''})$ is defined as the sequence of elementary transformations $t_{i1}, \ldots, t_{ik}, t_{ik+1}, \ldots, t_{ih}$ that defines the T-derivation $m_0, m_1, \ldots, m_{k-1}, m_k, m_{k+1}, \ldots, m_{h-1}, m_h$ from $m = m_0$ to $m'' = m_h$, where $t_{ij}(m_{j-1}) = m_j$ for $1 \leq j \leq h$. We say that $\oplus(T_{m,m'}, T_{m',m''})$ *takes* m *to* m'' *over* m'. We also write $\oplus(T_{m,m'}, T_{m',m''})$ as $T_{m,m'} \oplus T_{m',m''}$.

Property A_4. $\forall$ m, m', m'' $\in$ M: $\delta(m, m') \leq \delta(m, m'') + \delta(m'', m')$ (i.e., the triangle inequality)
Verification

It holds that $T_{m,m'} \oplus T_{m',m''} \in \boldsymbol{T_{m,m''}}$, but it is not implied that $T_{m,m'} \oplus T_{m',m''} \in \boldsymbol{ST_{m,m''}}$.
It also holds that $\varphi(T_{m,m'} \oplus T_{m',m''}) = h.c = \varphi(T_{m,m'}) + \varphi(T_{m',m''})$
$= \delta(m, m') + \delta(m', m'') = k.c + (h-k).c$.
Therefore, $\forall\, T_{m,m''} \in \boldsymbol{ST_{m,m''}}$:
$\delta(m, m'') = \varphi(T_{m,m''}) \leq h.c = \varphi(T_{m,m'}) + \varphi(T_{m',m''}) = \delta(m, m') + \delta(m', m'')$.

Property M_2 is the representation condition for ordinal measurement. It can be verified by referring to the observation criterion for the dissimilarity ordering $\bullet\geq$ on M.

Property M_2. $\forall\ m_1, m_2, m_3, m_4 \in M$: $(m_1, m_2) \bullet\geq (m_3, m_4) \Leftrightarrow \delta(m_1, m_2) \geq \delta(m_3, m_4)$ (i.e., representation condition for ordinal measurement)

Verification

$\Rightarrow$ *implication*

If $(m_1, m_2) \bullet\geq (m_3, m_4)$, then the length of the sequences in $\boldsymbol{ST_{m1,m2}}$ is at least as great as the length of the sequences in $\boldsymbol{ST_{m3,m4}}$. Suppose the length of the sequences in $\boldsymbol{ST_{m1,m2}}$ and $\boldsymbol{ST_{m3,m4}}$ is respectively k and k'. It then holds that $\delta(m_1, m_2) = k.c \geq k'.c = \delta(m_3, m_4)$.

$\Leftarrow$ *implication*

On the other hand, if $\delta(m_1, m_2) \geq \delta(m_3, m_4)$, then this implies that the length of the sequences in $\boldsymbol{ST_{m1,m2}}$ is at least as great as the length of the sequences in $\boldsymbol{ST_{m3,m4}}$. Hence, it is observed that $(m_1, m_2) \bullet\geq (m_3, m_4)$.

Property M_3 is the segmental additivity property. It is satisfied because of the specific construction of the proximity structure by means of a constructively complete and inverse constructively complete set of elementary transformation types.

Property M_3. $\forall\ m, m', m'' \in M$: $<mm''m'> \Leftrightarrow \delta(m, m') = \delta(m, m'') + \delta(m'', m')$ (i.e., segmental additivity)

Verification

$\Rightarrow$ *implication*

$<mm''m'>$

$\Rightarrow \exists\ T_{m,m'} \in \boldsymbol{ST_{m,m'}}$ that defines a T-derivation $m_0, m_1, \ldots, m_i, \ldots, m_{k-1}, m_k$, with $m = m_0$ and $m' = m_k$ and $m'' = m_i$ $(0 \leq i \leq k)$

(if m'' would not lie on a T-derivation defined by some shortest sequence of elementary transformations that takes m to m', then $<mm''m'>$ cannot hold as there would exist a $m_i \neq m''$ on a T-derivation defined by a shortest sequence of elementary transformations from m to m' such that $(m, m'') \bullet\geq (m, m_i)$ and $(m, m') \bullet\geq (m, m')$, but not $(m_i, m') \bullet\geq (m'', m')$.)

$\Rightarrow \forall\ i \in \{0, 1, \ldots, k\}$: $\delta(m, m') = \delta(m, m_i) + \delta(m_i, m')$

(Because of the triangle inequality we have $\delta(m, m') \leq \delta(m, m_i) + \delta(m_i, m')$. But if for some m_i it holds that $\delta(m, m') < \delta(m, m_i) + \delta(m_i, m')$, then m_i cannot be an abstraction of a T-derivation defined by a <u>shortest</u> sequence of elementary transformations that takes m to m'.)

$\Rightarrow \delta(m, m') = \delta(m, m'') + \delta(m'', m')$

$\Leftarrow$ implication

$\delta(m, m') = \delta(m, m'') + \delta(m'', m')$

$\Rightarrow \exists\ T_{m,m'} \in \boldsymbol{ST_{m,m'}}$ that defines a T-derivation $m_0, m_1, \ldots, m_i, \ldots, m_{k-1}, m_k$, with $m = m_0$ and $m' = m_k$ and $m'' = m_i$ $(0 \leq i \leq k)$

$\Rightarrow <mm''m'>$

Property M_4 is not part of the representation condition for a metric with additive segments. It is the uniqueness theorem associated with the representation.

Property M_4. If δ' is another metric on M satisfying the above conditions, then there exist $\beta > 0$ such that $\delta' = \beta\delta$.

Verification

For all $m, m' \in M$, assume $\delta(m, m') = \varphi(T_{m,m'}) = k.c$, where k is the length of $T_{m,m'} \in \boldsymbol{ST_{m,m'}}$ and c is a positive real number. Let c' be any other positive real number. Then, $\delta'(m, m') = \varphi(T_{m,m'}) = k.c'$. Both δ and δ' satisfy M_1, M_2 and M_3. For all $m, m' \in M$ it holds that $\delta'(m, m') = (c'/c).\delta(m, m') = \beta.\delta(m, m')$. As $c' > 0$ and $c > 0$, it holds that $\beta > 0$.